Universidad Nacional Autónoma de México

Facultad de Ciencias

Ajuste de Datos por Interpolación Bivariada de Akima para Fluidos de Yacimientos Petroleros

TESIS

QUE PARA OBTENER EL TíTULO DE:

Actuario

PRESENTA:

Isaí López Servín

DIRECTOR DE TESIS:
M. en C. César Carreón Otañez

Facultad de Ciencias, 2018
CIUDAD UNIVERSITARIA, CD.MX.

Dedicado a
Dios, a mi familia y a Jacqueline

Agradecimientos

Agradezco a Dios por hacer posible la culminación de este trabajo a pesar de las dificultades.

A mi familia, a aquellos que me apoyaron incondicionalmente y los que disentían.

A mi asesor por dirigir este trabajo hasta su culminación.

A los sinodales por sus aportaciones y enriquecer con ellas este trabajo.

Índice general

Resumen

La interpolación consiste en estimar valores desconocidos en una región a partir de datos iniciales. Se tienen dos escenarios para el planteamiento del problema: cuando los datos provienen de una función desconocida y que se aproxima por una nueva función o cuando son una lista de puntos y se requiere información que no se tiene.

En ambos casos los métodos de interpolación usan funciones polinomiales para aproximar a la función desconocida o crear una para los datos que se buscan.

En los yacimientos petroleros existen distintos tipos de fluidos que surgen como resultado del proceso de extracción, para la obtención de hidrocarburos necesarios principalmente para combustible y materia prima para plásticos. Los fluidos dentro del yacimiento están sujetos a una presión y temperatura con propiedades específicas a cada componente.

Dentro del proceso de extracción las condiciones iniciales se ven modificadas, con lo que sus propiedades varian debido al cambio de presión y temperatura. Los datos se conforman de una muestra discreta y para cada tipo de mezcla son usados en otra clase de procesos. Debido a ello se requiere información que con la que no se cuenta, es ahí, donde nace la necesidad de interpolar, de proponer valores que regiones desconocidas.

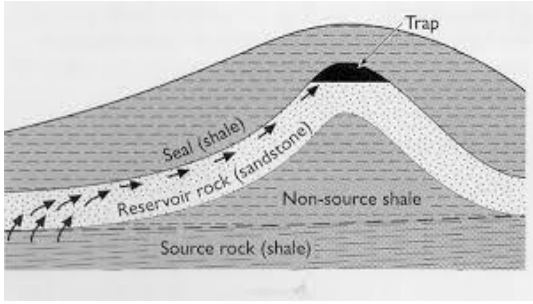
Los métodos presentados en la tesis son una revisión de los métodos clásicos unidimensionales (Newton, Lagrange, Hermite, Spline Cúbico y Hermite Cúbico), con los cuales se propuso una metodología para usarlos en el caso de datos bivariados, además de incluir otro que por si sólo está pensando para funciones de dos variables desarrollado por Akima [4].

Finalmente, son programados en una interfaz gráfica en el software numérico Matlab para realizar un análisis de los datos. Proponer un método con base en su rapidez y forma en que conserva la estructura de los datos originales.

Capítulo 1

Yacimientos

El petróleo y el gas natural que se producen a partir de los campos de petróleo y gas que residen en rocas permeables y porosas (reservorios) en las cuales estos líquidos se han acumulado y colectado lo largo de extensión del tiempo geológico.
Los campos de petróleo y gas son caracterizados por cuatro elementos de geológico: (1) rocas de origen (source rocks) (2) rocas de reservorio (reservoir rocks), (3) sellos (seals) y (4) trampas (traps).



Rocas de Origen
Son rocas sedimentarias que fueron depositadas en aguas muy tranquilas, generalmente en pantanos inmóviles en tierra, bahías marinas poco profundas y tranquilas, o en entornos submarinos profundos.Las rocas de origen son compuestas por fragmentos minerales muy pequeños, entre los fragmentos se encuentran los restos de material orgánico, generalmente algas, pequeños fragmentos de madera o piezas de la materia blanda y partes de plantas terrestres, cuando estos sedimentos de grano fino son enterrados por sedimentos posteriores y el aumento del calor y presión los sedimentos blandos se convierten en estratos de roca dura.
Si se produce un nuevo entierro, entonces la temperatura sigue aumentando, cuando las temperatura de superan los 120^oC las rocas comienza a generar el

petróleo y el gas natural, para dicho proceso se necesitan millones.

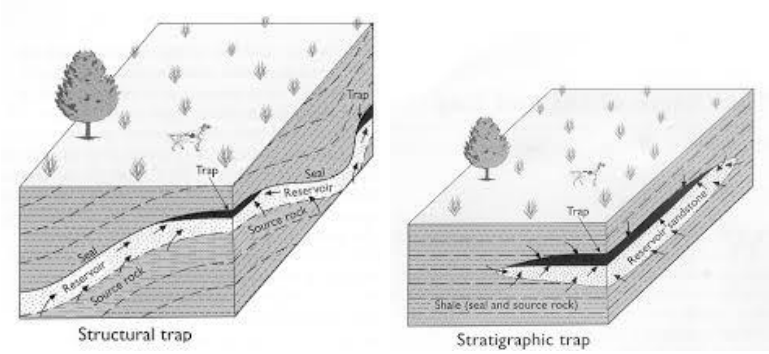
Rocas de reservorio
Las rocas de reservorio de petróleo y gas son porosas y permeables, contienen pasillos interconectados de poros microscópicos u orificios que ocupan las áreas entre los granos minerales de la roca, cuando el petróleo y el gas han sido expulsados naturalmente desde las rocas de origen, la mayor parte entran o migran hacia rocas adyacentes del embalse.
Una vez que el petróleo y el gas ingresan a la roca del yacimiento, son relativamente libres de moverse, más las rocas del yacimiento están inicialmente saturadas con agua salina subterránea.

Sellos
Los sellos generalmente son rocas de grano muy fino sin poros, espacios o poros demasiado pequeños para permitir la entrada de fluidos.

Trampa
El petróleo y el gas natural continúan migrando a través de espacios de los poros de la roca del yacimiento hasta que se bloqueen todos los movimientos del petróleo y el gas debido a su disposición física de uno o más sellos, un arreglo del reservorio y los sellos se llama trampa.
Existen dos tipos principales de trampas, estructurales y estratigráficas:

1. Estructurales: Se forman cuando la roca del yacimiento y el sello subyacente se han deformado, por lo general, esta deformación tiene lugar decenas de millones o cientos de millones de años después de la deposición de los sedimentos que se convierten en sellos y rocas de reservorio, el petróleo y el gas migran hacia arriba a través del reservorio y se acumulan en la parte más alta de la estructura.

2. Estratigráficos: se forman cuando la roca reservorio se deposita como un capa discontinua, los sellos se depositan al lado y encima del reservorio, el resultado es un reservorio de arenisca porosa rodeado de sellos.



En el campo de la explotación de yacimientos de petróleo, así como en los procesos de extracción, producción, acondicionamiento y transporte de los fluidos encontrados en tales yacimientos, los datos volumétricos y de comportamiento de fases correspondientes al fluido extraído son de esencial importancia para el manejo apropiado de los pozos petroleros así como para el correcto dimensionamiento de equipos y procesos.

La información del comportamiento del fluido es utilizada para evaluar las reservas de un yacimiento, para generar un plan óptimo de desarrollo y explotación así como para determinar la cantidad y la calidad de los fluidos producidos. También es utilizada para dimensionar, diseñar, simular y construir instalaciones apropiadas para el manejo del mismo.

Sin embargo, para poder realizar estas actividades, es necesario contar con información detallada a distintas condiciones de Temperatura y Presión relativa al comportamiento del fluido de interés, para el cual se está dimensionando, diseñando y simulando los equipos e instalaciones que lo procesarán. Esta información, contiene datos del comportamiento de fases, es decir, cuantas fases fluidas y en qué cantidad estarán presentes a una condición de Temperatura y Presión dada, así como las propiedades termodinámicas (densidad, derivadas de la densidad respecto a la temperatura y a la presión, capacidad calorífica, entalpía, entropía) y de transporte (viscosidad, conductividad térmica) para cada fase encontrada. Los datos son obtenidos a partir de un modelo semi-empírico, denominado ecuación de estado (EOS por sus siglas en ingles), modelo que previamente ha sido ajustado a los datos experimentales disponibles del fluido de interés.

La ecuación de estado (EOS) tiene las ventajas de poder reproducir los datos experimentales de manera satisfactoria y de poder predecir propiedades donde no existen datos experimentales disponibles. La mayor desventaja es el cos-

to computacional y de tiempo, necesario para poder resolver el modelo cada vez que se requiere la información del fluido de interés a una Temperatura y Presión definidas, en particular cuando se realizan cálculos o simulaciones de equipos o procesos en los que las variables Temperatura y Presión cambian fuertemente. En estos casos, los cálculos o simulaciones de equipos pueden tomar tiempos considerables, que pueden ir desde unas horas hasta varios días. La forma más utilizada de abordar el problema, es generar una tabla de propiedades, esto es, para una ventana con intervalos de Temperatura y de Presión definidas, se resuelve la ecuación de estado un número predeterminado de veces. De esta manera se obtienen propiedades en intervalos donde se espera que el equipo o proceso opere y la información está contenida en una tabla, guardada en un archivo digital. Dicha tabla tiene dos variables independientes, la Temperatura y la Presión, y cuando se especifican, se pueden obtener las propiedades previamente calculadas para todas las fases presentes.

Por la forma en que está dispuesta la información en la tabla, solo se cuenta con información de propiedades a temperaturas y presiones definidas, y si se requiere información a condiciones distintas, es necesario utilizar métodos de interpolación para obtener las propiedades a las condiciones de interés. Estos procedimientos o metodologías de interpolación deben tener las siguientes características:

1. Ejecutarse en un tiempo razonable.

2. Ejecutarse muchas muchas veces sin perdida de rapidez.

3. Respetar la topología de la variable a ser estudiada.

Las propiedades contenidas en la tabla, si bien son continuas, es decir, existe un valor de cada propiedad a cada temperatura y presión en las que fueron calculadas, pueden presentar comportamientos donde la derivada de la propiedad, ya sea en función de la temperatura o de la presión, no son continuas. Este fenómeno es causado por un cambio de fase del fluido de interés en ciertas regiones de temperatura y presión. Esta situación hace que la correcta selección del método de interpolación sea de fundamental importancia.

El ajuste y la selección del método se realiza sobre superficies generadas por el comportamiento de la medida de cada propiedad que proporcionan los datos.

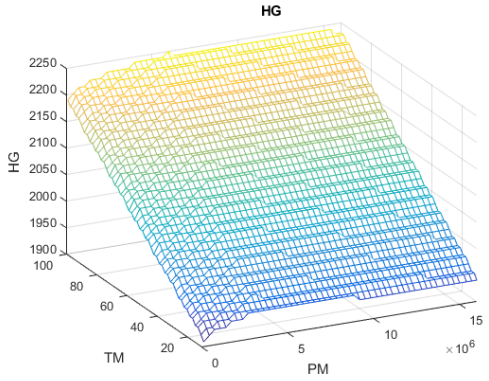


Figura 1.1: Propiedad de fluido

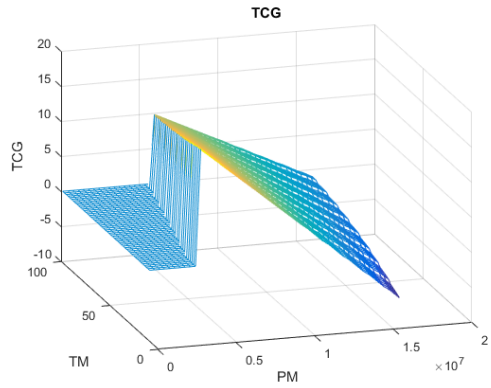


Figura 1.2: Propiedad de fluido

La interpolación lineal es el método base utilizado para aproximar valores en las propiedades, sin embargo, existen métodos de interpolación con mejor eficiencia para estimar valores.

El propósito del trabajo, es el analizar distintos métodos alternativos al interpolante lineal, con objetivo de proponer nuevos modelos que ajusten mejor los datos a las propiedades de los fluidos, logrando estimar los datos sobre superficies más suaves, ya que los interpolantes de mayor complejidad, contemplan más supuestos que pueden ayudar a un mejor ajuste, además, se desarrollarán programas computacionales en el software numérico MATLAB para los distintos métodos que se estudiarán.

En el primer capítulo del trabajo, se darán algunos conceptos sobre yacimientos de petróleo, las definiciones sólo se tratarán de manera muy general, ya que se enfocará en el desarrollo de métodos numéricos para interpolación.

En el segundo capítulo se desarrollará la construcción de los interpolantes univariados, donde se estudiarán los clásicos de Lagrange, Newton, Hermite, Spline Cúbico y Hermite Cúbico.

El capítulo tres estudiará métodos de interpolación bivariados, primero se hará un acercamiento, interpolando múltiples veces mediante métodos univariados, posteriormente se explicará la construcción del Interpolante Bivariado de Akima.

En el capítulo cuatro se describirán cada uno de los programas y la forma de uso de la interfaz gráfica donde se implementaron los distintos métodos descritos en los capítulos anteriores.

Finalmente en el capítulo cinco tratará sobre el análisis de algunos resultados obtenidos al aplicar la interpolación bivarida a funciones analíticas y a las propiedades de algunos fluidos en yacimientos de petróleos, para determinar el interpolante que mejor ajuste según las condiciones requeridas en la interpolación de los fluidos.

1.1. Definición básica de Yacimiento

Podemos entender el concepto de yacimiento como una unidad geológica de un volumen limitado, poroso y permeable que contiene hidrocarburos en estado líquido o gaseoso, estos yacimientos se definen en cinco elementos básicos

1. Fuente
2. Camino migratorio
3. Trampa
4. Almacenaje
5. Permeabilidad

1.2. Clasificación según el estado de los fluidos

1. Petróleo Negro: Se caracteriza por tener una gran cantidad de especies químicas, que contienen moléculas grandes y pesadas, además de no ser volátil en ese estado, teniendo un color negro o verduzco.
2. Petróleo Volátil: Tiene la particularidad, que al sufrir una pequeña reducción en la presión el fluido puede desprender grandes cantidades de gas, su color puede variar entre marrón y naranja.
3. Gas Condensado: Para este caso, a medida que la presión decrece el gas se condensa formando un líquido dentro del yacimiento, sus colores característicos pueden ser marrón, naranja, verduzco transparente.

4. Gas Húmedo: Predominan moléculas pequeñas y pesadas, su temperatura está por debajo a la del yacimiento, presenta líquido pero en la superficie, su color es transparente.

5. Gas Seco: Principalmente formado por metano, este tipo de gas tiene moléculas más ligeras en comparación al los gases húmedos, presenta mezcla de hidrocarburos dentro y en la superficie del yacimiento, sin embargo, no presenta líquidos dentro ni fuera del yacimiento.

6. Asfalténicos: Cuenta con composición química constante, sin embargo, cuando la presión del yacimiento baja, el gas puede producirse más fácilmente, al aumentar la presión, se condensa y el líquido generado queda atrapado dentro de la roca del yacimiento.

1.3. Clasificación geológica de yacimientos

1. Estratigráficos: Formada por piedras calizas o dolomitas porosas, además de tener cambios en la permeabilidad.

2. Estructurales: Contiene rocas ígneas con domos salinos.

3. Combinado: Presentan combinaciones de los dos antes mencionados.

1.4. Clasificación respecto al Punto Burbuja

Entenderemos por punto burbuja, a las condiciones de presión y temperatura tal que aparezca la primera burbuja de gas en el hidrocarburo.

1. Subsaturados: Yacimientos cuya presión está por debajo del punto burbuja, se presenta en fase líquida y las burbujas de gas se desprenden una vez alcanzado el punto burbuja.

2. Saturados: La presión inicial del yacimiento es menor al punto burbuja, consta de una zona líquida y una zona gaseosa, la zona líquida se encuentra en su punto burbuja.

1.5. Clasificación de acuerdo al mecanismo de producción

1. Producción primaria: En un principio la producción de hidrocarburos se realiza mediante la energía natural, en este tipo de producción el petróleo y el gas son desplazados hacia pozos productores mediante:

   a) Expansión de fluido
   b) Desplazamiento de fluidos.
   c) Drenaje gravitacional.
   d) Expulsión Capilar

2. Producción secundaria: En este caso, para lograr producción es necesario usar gas natural o inyección de agua, con el propósito de mantener la presión en el yacimiento.

Normalmente se usan ambos métodos para la producción de hidrocarburos.

1.6. Clasificación de acuerdo al volumen

1. Volumétricos: Cuando no existe un acuífero adyacente al yacimientos.
2. No volumétricos: El volumen disponible de hidrocarburos, se reduce debido a la intrusión de agua de un acuífero adyacente.

Capítulo 2

Métodos de Interpolación Univariado

En el capítulo se darán a conocer distintas propuestas de metodologías para realizar una interpolación, método que consiste en construir una curva a partir de una serie de datos finita, la historia nos muestra las primeras formas de uso del método de interpolación para resolver ciertas necesidades de la época.

Los primeros vestigios históricos que se tienen sobre un uso muy rudimentario de la interpolación se ubican aproximadamente en el año 1,700 A.C. en la civilización babilónica, donde usaban una recta entre dos puntos para aproximar determinados valores, un ejemplo de ello era la aproximación a potencias. En la actualidad gracias a los avances en la computación, no resulta nada difícil aproximar el valor de la variable t de la igualdad $((2+(1/10)))^t = 4$, ya que solo basta con calcular $t = ln(4)/ln(2.1) \approx 1.386294/0.7419373 \approx 1.868479$ donde el valor de los logaritmos fácilmente obtenidos en una calculadora científica, sin embargo los babilónicos no contaban con tal tecnología, por tanto ellos aproximaban el valor de t de manera gráfica, pasando una recta entre un punto mayor y menor al valor que se necesitaba calcular y daban una aproximación observando el valor en la recta, a esta forma de interpolar se conoce como "interpolación lineal"siendo una de las ideas más intuitivas para resolver el problema, ahora analicemos como se aproxima el valor de t mediante este método, primero veamos $((2 + (1/10)))^t = 4$ junto a t como pareja ordenada $(t, 4)$, ahora busquemos valores t_1 y t_2 tales que $y_1 < y = 4 < y_2$, tomemos $t_1 = 1$ y $t_2 = 2$ entonces tenemos $y_1 = 2.1 < y = 4 < y_2 = 4.41$, graficando y uniendo ambos puntos por una recta tenemos la siguiente figura

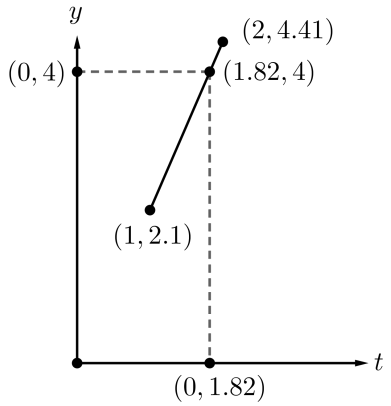


Figura 2.1

podemos observar que la aproximación es $t \approx 1.82$ que no difiere mucho del resultado aproximado a través de logaritmos. Sin embargo al pasar de los años, la necesidad de poder aproximar valores de funciones desconocidas, con mejores métodos que una simple recta, hizo que se buscarán nuevas formas de interpolación.

En este capítulo estudiaremos algunos métodos de interpolación, clasificando los interpolantes en dos partes, por un lado, analizaremos los interpolantes cuya característica es ser un solo polinomio $P(x)$ que pase por una serie de puntos (x_i, y_i) con $i = 0, 1, ..., n$ cumpliendo la condición $P(x_i) = y_i$, por otro lado estudiaremos los interpolantes polinomiales a trozos, que consisten en pasar una serie de polinomios f_i en cada subintervalo $[x_i, x_{i+1}]$ y además cumplen que $f_i(x_i) = y_i$ y $f_i(x_{i+1}) = y_{i+1}$ para $i = 0, 1, ..., n - 1$.

2.1. Interpolación Polinomial

En esta sección, describiremos métodos clásicos para encontrar un polinomio, que pase por una serie de datos (x_i, y_i) para $i = 0, 1, 2, ..., n$.

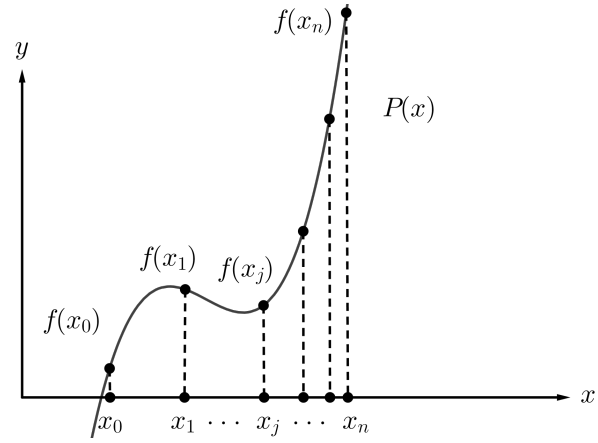


Figura 2.2

2.1.1. Interpolación Polinomial de Lagrange

Dada una serie de puntos distintos $(x_0, y_0), ..., (x_n, y_n)$ suponiendo una función f desconocida que cumple $f(x_i) = y_i$, se construirá un polinomio que pase por cada uno de los puntos dados, aproximando una función f mediante un polinomio de grado n, de tal manera que se cumpla que $P(x_0) = y_0, ..., P(x_n) = y_n$.

Para obtener el polinomio que satisface lo anterior mencionado, se usará la metodología descrita en el siguiente teorema.

Teorema 2.1 *Sea $x_0, x_1, x_2, ..., x_n$ $n+1$ valores distintos, si f es una función que está definida en esos números, entonces existe un polinomio único $P(x)$ de un grado no mayor a n que cumple la siguiente propiedad:*

$$P(x_j) = f(x_j) \qquad \text{con } j = 0, 1, 2, ..., n$$

Donde $P(x)$ se define como:

$$P(x) = f(x_0)L_{n,0}(x) + f(x_1)L_{n,1}(x) + \cdots + f(x_n)L_{n,n}(x) = \sum_{j=0}^{n} f(x_j)L_{n,j}(x)$$

Y $L_{n,j}(x)$ *está definido por:*

$$L_{n,j}(x) = \frac{(x-x_0)(x-x_1)\cdots(x-x_{j-1})(x-x_{j+1})\cdots(x-x_n)}{(x_j-x_0)(x_j-x_1)\cdots(x_j-x_{j-1})(x_j-x_{j+1})\cdots(x_j-x_n)} = \prod_{\substack{i=0 \\ i\neq j}}^{n} \frac{x-x_i}{x_j-x_i}$$

Para $j = 0, 1, 2, ..., n$.

Al polinomio anterior se le conoce como n-ésimo polinomio Lagrange.

[2, pág. 109]

2.1.2. Interpolación Polinomial de Newton

Podemos notar que el interpolante de Lagrange tiene un costo numérico alto dada su estructura y la construcción del mismo, con el propósito de hacer más eficiente el cálculo, surge una segunda opción, construir un polinomio interpolante mediante el método de "diferencias divididas de Newton", que es posible obtener recursivamente, encontrando los coeficientes de polinomio de manera más eficiente, consideremos a $P_n(x)$ el n-ésimo polinomio interpolante de Lagrange y f una función definida para cada uno de los puntos $x_0, x_1, x_2, ..., x_n$ que cumple $f(x_0) = y_0, f(x_1) = y_1, f(x_2) = y_2 ..., f(x_n) = y_n$, expresemos a $P_n(x)$ como

$$P_n(x) = \alpha_0 + \alpha_1(x-x_0) + \alpha_2(x-x_0)(x-x_1) + \cdots + \alpha_n(x-x_0)\cdots(x-x_{n-1})$$

sabiendo que se debe cumplir que $P_n(x_i) = f(x_i) = y_i$ para $i = 0, 1, 2, ..., n$ podemos encontrar los valores α_i,

$$\alpha_0 = P_n(x_0) = f(x_0)$$

$f(x_0) + \alpha_1(x_1 - x_0) = P_n(x_1) = f(x_1)$ y despejando tenemos

$$\alpha_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

a este procedimientos se le llama "diferencias divididas", en el caso de $\alpha_0 = f(x_0)$ se le conoce como *diferencia dividida cero de f respecto a* x_0 y se denota como $f[x_0] = f(x_0)$ y para cada uno de los puntos x_i tenemos $f[x_i] = f(x_i)$, a α_1 se le conoce como la *primera diferencia dividida de f respecto a* x_1 *y* x_0 y se denota

$$\alpha_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0} = f[x_0, x_1]$$

y en general para x_i

$$f[x_i, x_{i+1}] = \frac{f[x_{i+1}] - f[x_i]}{x_{i+1} - x_i}$$

de manera recursiva

$$f[x_i, x_{i+1}, x_{i+2}] = \frac{f[x_{i+1}, x_{i+2}] - f[x_i, x_{i+1}]}{x_{i+2} - x_i}$$

segunda diferencia dividida

$$f[x_i, x_{i+1}, x_{i+2}, x_{i+3}] = \frac{f[x_{i+1}, x_{i+2}, x_{i+3}] - f[x_i, x_{i+1}, x_{i+2}]}{x_{i+3} - x_i}$$

tercera diferencia dividida

$$\vdots$$

$$f[x_i, x_{i+1}, x_{i+2}, ..., x_{i+j}] = \frac{f[x_{i+1}, x_{i+2}, ..., x_{i+j}] - f[x_i, x_{i+1}, ..., x_{i+j-1}]}{x_{i+j} - x_i}$$

j-ésima diferencia dividida

se definen los coeficientes α_i con $i = 0, 1, 2, ..., n$ como sigue

$$\alpha_j = f[x_0, x_1, x_2, ..., x_j]$$

por lo tanto el polinomio $P_n(x)$ se puede expresar

$$P_n(x) = f[x_0] + \sum_{j=1}^{n} f[x_0, x_1, x_2, ..., x_j](x - x_0)(x - x_1) \cdots (x - x_{j-1})$$

conocido como el polinomio interpolante de Newton.

[2, pág. 122-124].

2.1.3. Interpolación Polinomial de Hermite

Los polinomios interpolantes de Newton y Lagrange, tienen la desventaja, de presentar curvas muy pronunciadas en algunos subintervalos (véase las figuras 2.3 y 2.4), lo cual podría causar errores numéricos grandes al estimar los valores intermedios de cada $[x_i, x_{i+1}]$, por tal motivo, se propone un polinomio interpolante de grado $2n + 1$, que coincide con los $n + 1$, $(x_0, y_0), (x_1, y_1), ..., (x_n, y_n)$, además, el interpolante contempla el valor de f' para cada uno de los puntos, con el propósito de construir un polinomio que ajuste de mejor manera a los puntos (x_i, y_i) y disminuir las curvas muy pronunciadas para atenuar los posibles errores en valores intermedios $[x_i, x_{i+1}]$, a dicho polinomio denotado $H_{2n+1}(x)$ se le conoce como *Polinomio de Hermite* y se define en el siguiente teorema.

Polinomio de Hermite

Teorema 2.2 *Si $f \in C^1[a,b]$ y siendo $x_0, x_1, x_2, ..., x_n$ números distintos, el polinomio de mínimo grado que interpola f y f' en $x_0, x_1, x_2, ..., x_n$ es el polinomio de Hermite de grado $2n+1$ dado por*

$$H_{2n+1}(x) = \sum_{j=0}^{n} f(x_j) H_{n,j}(x) + \sum_{j=0}^{n} f'(x_j) \hat{H}_{n,j}(x)$$

donde

$$H_{n,j}(x) = [1 - 2(x - x_j) L'_{n,j}(x_j)] L^2_{n,j}(x)$$

$$\hat{H}_{n,j}(x) = (x - x_j) L^2_{n,j}(x)$$

[2, pág. 134].

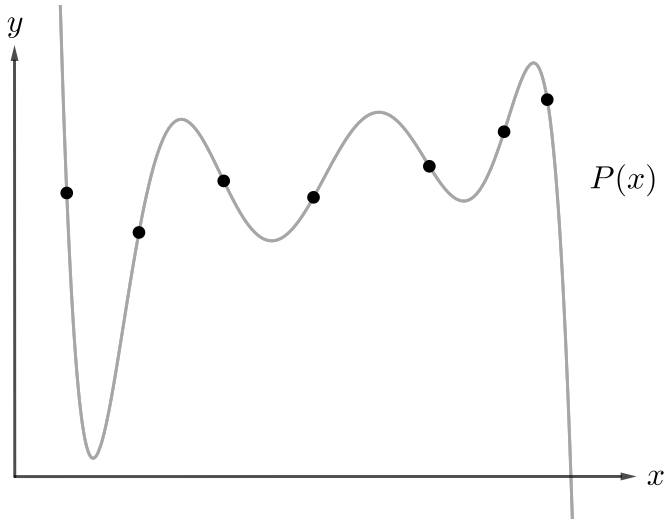


Figura 2.3: Polinomio interpolante de Newton.

Diferencias divididas en el Polinomio de Hermite

Con la finalidad de poder implementar el interpolante de Hermite de una manera más práctica, es posible llegar al polinomio mediante las diferencias divididas de Newton, lo cual reduce el trabajo de calcular los polinomios de Lagrange y su respectiva derivada.

Siendo $x_0, x_1, x_2, ..., x_n$ números distintos y definidos en f y f', sean los puntos $z_0, z_1, z_2, ..., z_{2n+1}$ de la siguiente manera

$$z_{2i} = z_{2i+1} = x_i \text{ para } i = 0, 1, 2, ..., n$$

notemos que $f[z_{2i}, z_{2i+1}]$ no tiene un valor asignado, para resolverlo podemos dar las siguiente aproximación

$$f[z_{2i}, z_{2i+1}] \approx \lim_{h \to 0} \frac{f(x_i + h) - f(x_i)}{h} = f'(x_i)$$

para $i = 0, 1, 2, ..., n$.

Entonces, la nueva expresión del polinomio de Hermite de grado $2n + 1$ será

$$H_{2n+1}(x) = f[z_0] + \sum_{j=1}^{2n+1} f[z_0, z_1, ..., z_j](x - z_0)(x - z_1), \cdots, (x - z_{j-1})$$

[2, pág. 137-138].

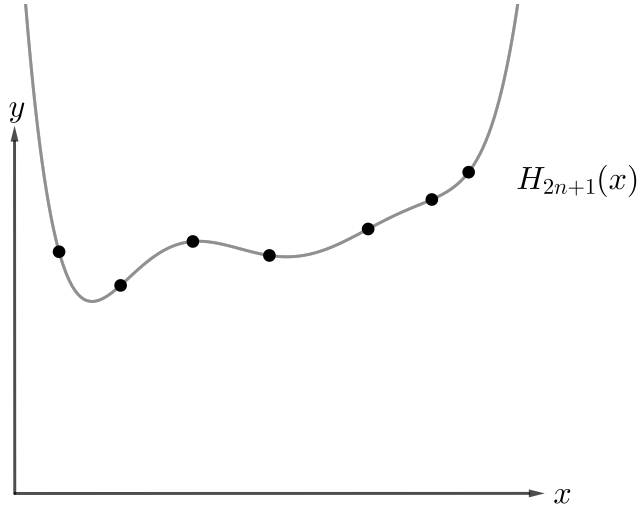


Figura 2.4: Polinomio interpolante de Hermite.

2.2. Interpolación Polinomial a Trozos

En la sección anterior se estudiaron los Polinomios Interpolantes de Lagrange, Newton y Hermite, en los cuales el inconveniente es el grado del polinomio generado, si el número de puntos (x_i, y_i) aumenta, entonces también crecerá el grado del mismo apareciendo números que rebasan por mucho la escala de los puntos interpolados, generando una mala estimación de valores intermedios como se muestra en la imagen 2.5.

Con el propósito de disminuir curvaturas tan grandes y así optimizar la estimación, los siguientes interpolantes se construirán a través de polinomios en cada subintervalo formado por los puntos $x_0, x_1, x_2, ..., x_n$.

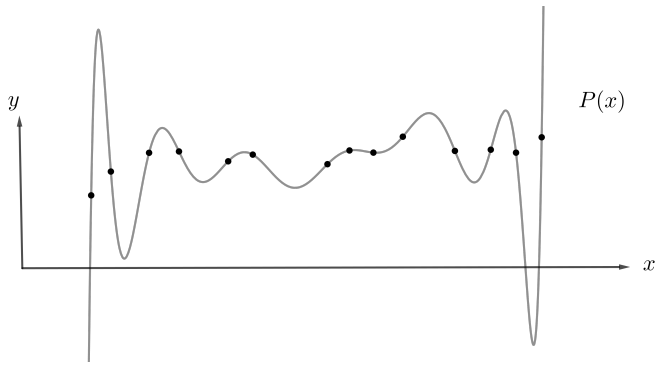


Figura 2.5: Desventaja del polinomio interpolante de Lagrange, Newton y Hermite.

Antes de comenzar a analizar algunos de ellos, veamos los conceptos que se usarán a lo largo de la sección.

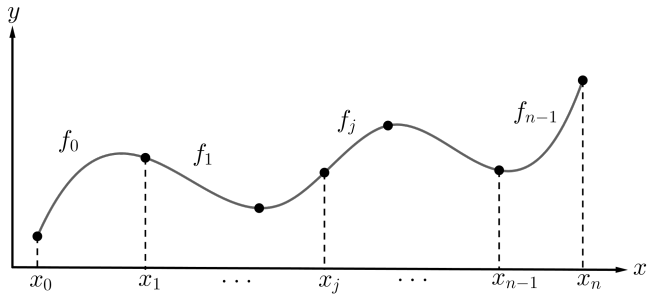


Figura 2.6

Si tenemos $(x_0, y_0), (x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$ $n+1$ puntos en $\mathbb{R}^2$ con $x_0 < x_1 < x_2 < \cdots < x_n$, definimos lo siguiente

$$t = x - x_i \text{ para } x \in [x_i, x_{i+1}]$$

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

$$s_i = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} = \frac{y_{i+1} - y_i}{h_i}$$

también a

$$f_i(x) \text{ con } x \in [x_i, x_{i+1}]$$

como el i-ésimo polinomio en el intervalo $[x_i, x_{i+1}]$, para $i = 0, 1, 2, ..., n-1$.

2.2.1. Interpolación Lineal

Como primer ejemplo se tiene el interpolante que consiste en unir segmentos de rectas para cada subintervalo de la partición, para lograrlo basta con encontrar la ecuación de la recta entre dos puntos:

$$\frac{f_i(x) - y_i}{y_{i+1} - y_i} = \frac{x - x_i}{x_{i+1} - x_i}$$

$$f_i(x) = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}(x - x_i) + y_i$$

$$f_i(x) = s_i \cdot t + y_i$$

para $x \in [x_i, x_{i+1}]$ e $i = 0, 1, 2, ..., n-1$, es importante resaltar que el interpolante cumple

$$f_i(x_{i+1}) = f_{i+1}(x_{i+1})$$

para $i = 0, 1, 2, ..., n-1$.

2.2.2. Interpolación Spline Cúbico

Con el propósito de encontrar mejores aproximaciones sobre funciones más suaves contemplando supuestos que puedan mejorar la estimación de los datos, se propone un método de interpolación conocido como Spline.

El método de interpolación consiste en construir un polinomio de grado tres por cada dos puntos consecutivos (x_i, y_i) para $i = 0, 1, 2, ..., n$, dichos polinomios deben cumplir las siguientes condiciones:

1. $f_i(x) = \alpha_i + \beta_i(x - x_i) + \gamma_i(x - x_i)^2 + \delta_i(x - x_i)^3$ para $i = 0, 1, 2, ..., n-1$
2. $f_i(x_i) = f(x_i) = y_i$ y $f_i(x_{i+1}) = f(x_{i+1}) = y_{i+1}$ para $i = 0, 1, 2, ..., n-1$
3. $f_i(x_{i+1}) = f_{i+1}(x_{i+1})$ para $i = 0, 1, 2, ..., n-1$

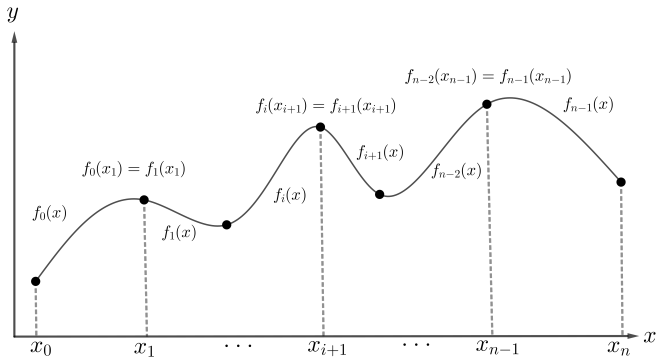


Figura 2.7

4. $f_i'(x_{i+1}) = f_{i+1}'(x_{i+1})$ para $i = 0, 1, 2, ..., n-1$

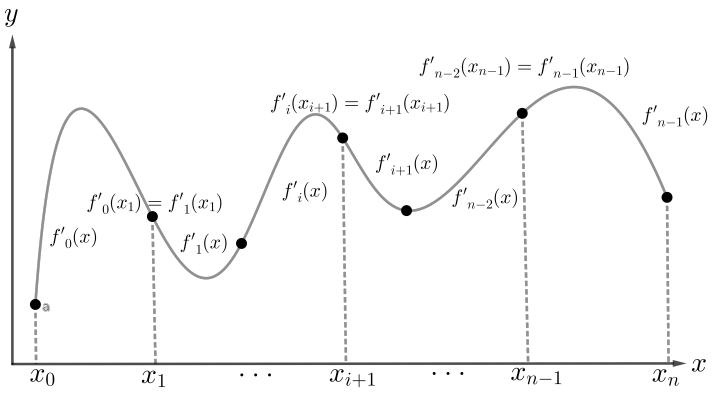


Figura 2.8

5. $f_i''(x_{i+1}) = f_{i+1}''(x_{i+1})$ para $i = 0, 1, 2, ..., n-1$

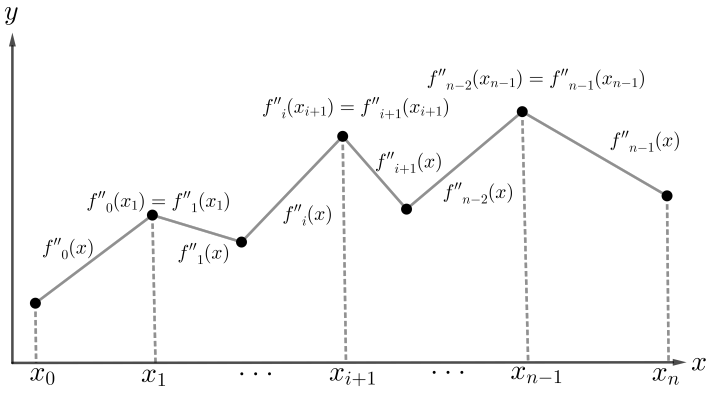


Figura 2.9

Ahora la tarea consiste en encontrar los términos adecuados α_i, β_i, γ_i, δ_i de cada uno de los polinomios f_i para $i = 0, 1, 2, ..., n-1$ que cumplan con las condiciones anteriores.

Si consideramos evaluar $f_i(x_i)$ para $i = 0, 1, 2, ..., n-1$ obtenemos cada α_i

$$f_i(x_i) = \alpha_i + \beta_i(x_i - x_i) + \gamma_i(x_i - x_i)^2 + \delta_i(x_i - x_i)^3 = \alpha_i$$

$$\alpha_i = y_i$$

de la condición (2) obtenemos que

$$f_i(x_{i+1}) = \alpha_i + \beta_i(x_{i+1} - x_i) + \gamma_i(x_{i+1} - x_i)^2 + \delta_i(x_{i+1} - x_i)^3$$

$$f_{i+1}(x_{i+1}) = \alpha_{i+1} + \beta_{i+1}(x_{i+1} - x_{i+1}) + \gamma_{i+1}(x_{i+1} - x_{i+1})^2 + \delta_{i+1}(x_{i+1} - x_{i+1})^3$$

$$\alpha_{i+1} = \alpha_i + \beta_i(x_{i+1} - x_i) + \gamma_i(x_{i+1} - x_i)^2 + \delta_i(x_{i+1} - x_i)^3$$

$$\alpha_{i+1} = \alpha_i + \beta_i h_i + \gamma_i h_i^2 + \delta_i h_i^3 \tag{2.1}$$

Para poder encontrar una expresión β_i derivemos el polinomio f_i

$$f_i'(x) = \beta_i + 2\gamma_i(x - x_i) + 3\delta_i(x - x_i)^2$$

tomando en cuenta la condición (4)

$$f_i'(x_{i+1}) = \beta_i + 2\gamma_i(x_{i+1} - x_i) + 3\delta_i(x_{i+1} - x_i)^2$$

$$f_{i+1}'(x_{i+1}) = \beta_{i+1} + 2\gamma_{i+1}(x_{i+1} - x_{i+1}) + 3\delta_{i+1}(x_{i+1} - x_{i+1})^2$$

$$\beta_{i+1} = \beta_i + 2\gamma_i(x_{i+1} - x_i) + 3\delta_i(x_{i+1} - x_i)^2$$

$$\beta_{i+1} = \beta_i + 2\gamma_i h_i + 3\delta_i h_i^2 \tag{2.2}$$

Ahora calculando la segunda derivada del polinomio f_i y usando la condición (5) obtenemos

$$f_i''(x) = 2[\gamma_i + 3\delta_i(x - x_i)]$$

$$f_i''(x_{i+1}) = 2[\gamma_i + 3\delta_i(x_{i+1} - x_i)]$$

$$f_{i+1}''(x_{i+1}) = 2[\gamma_{i+1} + 3\delta_{i+1}(x_{i+1} - x_{i+1})]$$

$$2\gamma_{i+1} = 2[\gamma_i + 3\delta_i(x_{i+1} - x_i)] \tag{2.3}$$

$$\gamma_{i+1} = \gamma_i + 3\delta_i h_i \tag{2.4}$$

despejando a δ_i de la expresión 2.3 y sustituyendo en las ecuaciones anteriores 2.1 y 2.2

$$\delta_i = \frac{\gamma_{i+1} - \gamma_i}{3h_i}$$

$$\beta_{i+1} = \beta_i + h_i(\gamma_{i+1} + \gamma_i)$$

$$\alpha_{i+1} = \alpha_i + \beta_i h_i + \frac{h_i^2}{3}(2\gamma_i + \gamma_{i+1})$$

obteniendo a β_i de la última ecuación

$$\beta_i = \frac{1}{h_i}(\alpha_{i+1} - \alpha_i) - \frac{h_i}{3}(2\gamma_i + \gamma_{i+1}) \tag{2.5}$$

con este resultado podemos encontrar una expresión para β_{i-1}

$$\beta_{i-1} = \frac{1}{h_{i-1}}(\alpha_i - \alpha_{i-1}) - \frac{h_{i-1}}{3}(2\gamma_{i-1} + \gamma_i) \tag{2.6}$$

aplicando el mismo procedimiento y sustituyendo en la ecuación 2.2

$$\beta_{i+1} = \beta_i + h_i(\gamma_{i+1} + \gamma_i)$$

$$\beta_i = \beta_{i-1} + h_{i-1}(\gamma_i + \gamma_{i-1})$$

$$\frac{1}{h_i}(\alpha_{i+1}-\alpha_i)-\frac{h_i}{3}(2\gamma_i+\gamma_{i+1}) = \frac{1}{h_{i-1}}(\alpha_i-\alpha_{i-1})-\frac{h_{i-1}}{3}(2\gamma_{i-1}+\gamma_i)+h_{i-1}(\gamma_i+\gamma_{i-1})$$

$$h_i(2\gamma_i + \gamma_{i+1}) + h_{i-1}(2\gamma i + \gamma_{i-1}) = \frac{3}{h_i}(\alpha_{i+1} - \alpha_i) - \frac{3}{h_{i-1}}(\alpha_i - \alpha_{i-1})$$

$$h_{i-1}\gamma_{i-1} + 2(h_{i-1} + h_i)\gamma_i + h_i\gamma_{i+1} = \frac{3}{h_i}(\alpha_{i+1} - \alpha_i) - \frac{3}{h_{i-1}}(\alpha_i - \alpha_{i-1}) \tag{2.7}$$

para $i = 1, 2, ..., n-1$.

Todo el procedimiento anterior se realizó para construir un sistema de ecuaciones y encontrar el valor de γ_i ya que al obtener dichos valores podemos encontrar también los de β_i y δ_i. El proceso deja dos grados libres, los cuales son llamados *condiciones de frontera*, dependiendo de los valores elegidos es como se listan los diferentes tipos de Spline.

Spline Cúbico Natural

La condición de frontera para este Spline Cúbico es el siguiente

$$f_0''(x_0) = 0 \text{ y } f_{n-1}''(x_n) = 0$$

desarrollando para $f_0''(x_0)$ tenemos

$$f_0''(x_0) = 2[\gamma_0 + 3\delta_0(x_0 - x_0)] = 0$$

$$\gamma_0 = 0$$

y ahora para $f_{n-1}''(x_n)$

$$f''_{n-1}(x_n) = 2[\gamma_{n-1} + 3\delta_{n-1}(x_n - x_{n-1})] = 2[\gamma_{n-1} + 3\delta_{n-1}(h_{n-1})] = 0$$

usando la ecuación 2.3

$$2\gamma_n = 2[\gamma_{n-1} + 3\delta_{n-1}(h_{n-1})] = 0$$

$$\gamma_n = 0$$

con las dos ecuaciones obtenidas, el sistema generado por 2.7, se construye un sistema *tridiagonal* o *tipo banda*, para obtener los valores de γ_i, solo basta con resolver el sistema

$$Ax = b$$

donde

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & \cdots & 0 \\ h_0 & 2(h_0+h_1) & h_1 & \ddots & & & \vdots \\ 0 & h_1 & 2(h_1+h_2) & h_2 & \ddots & & 0 \\ 0 & 0 & h_2 & 2(h_2+h_3) & h_3 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & & \\ & & & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} & 0 \\ 0 & \cdots & \cdots & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$x = \begin{pmatrix} \gamma_0 \\ \gamma_1 \\ \gamma_2 \\ \\ \vdots \\ \\ \gamma_n \end{pmatrix}$$

$$b = \begin{pmatrix} 0 \\ \frac{3}{h_1}(\alpha_2 - \alpha_1) - \frac{3}{h_0}(\alpha_1 - \alpha_0) \\ \frac{3}{h_2}(\alpha_3 - \alpha_2) - \frac{3}{h_1}(\alpha_2 - \alpha_1) \\ \vdots \\ \\ \frac{3}{h_{n-1}}(\alpha_n - \alpha_{n-1}) - \frac{3}{h_{n-2}}(\alpha_{n-1} - \alpha_{n-2}) \\ 0 \end{pmatrix}$$

Spline Cúbico Completo

El interpolante tiene las condiciones de frontera dadas como :

$$f'_0(x_0) = f'(x_0) \text{ y } f'_{n-1}(x_n) = f'(x_n)$$

desarrollando $f'_0(x_0)$ tenemos

$$f'(x_0) = \beta_0 + 2\gamma_0(x_0 - x_0) + 3\delta_0(x_0 - x_0)^2$$

$$f'(x_0) = \beta_0$$

usando la ecuación 2.6

$$f'(x_0) = \frac{1}{h_0}(\alpha_1 - \alpha_0) - \frac{h_0}{3}(2\gamma_0 + \gamma_1)$$

$$2h_0\gamma_0 + h_0\gamma_1 = \frac{3}{h_0}(\alpha_1 - \alpha_0) - 3f'(x_0)$$

de manera similar con $f'_{n-1}(x_n)$

$$f'(x_n) = \beta_{n-1} + 2\gamma_{n-1}(x_n - x_{n-1}) + 3\delta_{n-1}(x_n - x_{n-1})^2 = \beta_n =$$

$$\beta_{n-1}+h_{n-1}(\gamma_n+\gamma_{n-1}) = \frac{1}{h_{n-1}}(\alpha_n-\alpha_{n-1})-\frac{h_{n-1}}{3}(2\gamma_{n-1}+\gamma_n)+h_{n-1}(\gamma_n+\gamma_{n-1})$$

$$h_{n-1}\gamma_{n-1} + 2h_{n-1}\gamma_n = 3f'(x_n) - \frac{3}{h_{n-1}}(\alpha_n - \alpha_{n-1})$$

ahora, veamos la nueva matriz y vector del sistema $Ax = b$

$$A = \begin{pmatrix} 2h_0 & h_0 & 0 & 0 & \cdots & \cdots & 0 \\ h_0 & 2(h_0+h_1) & h_1 & \ddots & & & \vdots \\ 0 & h_1 & 2(h_1+h_2) & h_2 & \ddots & & 0 \\ 0 & 0 & h_2 & 2(h_2+h_3) & h_3 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & & \\ & & & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} & 0 \\ 0 & \cdots & \cdots & 0 & 0 & h_{n-1} & 2h_{n-1} \end{pmatrix}$$

$$x = \begin{pmatrix} \gamma_0 \\ \gamma_1 \\ \gamma_2 \\ \\ \vdots \\ \\ \gamma_n \end{pmatrix}$$

$$b = \begin{pmatrix} \frac{3}{h_0}(\alpha_1 - \alpha_0) - 3f'(x_0) \\ \frac{3}{h_1}(\alpha_2 - \alpha_1) - \frac{3}{h_0}(\alpha_1 - \alpha_0) \\ \frac{3}{h_2}(\alpha_3 - \alpha_2) - \frac{3}{h_1}(\alpha_2 - \alpha_1) \\ \vdots \\ \frac{3}{h_{n-1}}(\alpha_n - \alpha_{n-1}) - \frac{3}{h_{n-2}}(\alpha_{n-1} - \alpha_{n-2}) \\ 3f'(x_n) - \frac{3}{h_{n-1}}(\alpha_n - \alpha_{n-1}) \end{pmatrix}$$

Spline Cúbico Not-A-Knot

Este Spline, en los puntos frontera, tiene la particularidad de contemplar un supuesto en la tercera derivada. Las condiciones de frontera son

$$f_0'''(x_0) = f_1'''(x_1) \text{ y } f_{n-2}'''(x_{n-1}) = f_{n-1}'''(x_n)$$

derivando obtenemos

$$f_i'''(x) = 6\delta_i$$

usando la ecuación 2.4 y sustituyendo en $f_0'''(x_0) = f_1'''(x_1)$

$$6\frac{\gamma_1 - \gamma_0}{3h_0} = 6\frac{\gamma_2 - \gamma_1}{3h_1}$$

$$h_1(\gamma_1 - \gamma_0) = h_0(\gamma_2 - \gamma_1)$$

$$h_1(\gamma_1 - \gamma_0) - h_0(\gamma_2 - \gamma_1) = 0$$

$$-h_1\gamma_0 + (h_0 + h_1)\gamma_1 - h_0\gamma_2 = 0$$

procediendo de la misma forma en $f_{n-2}'''(x_{n-1}) = f_{n-1}'''(x_n)$

$$6\frac{\gamma_{n-1} - \gamma_{n-2}}{3h_{n-2}} = 6\frac{\gamma_n - \gamma_{n-1}}{3h_{n-1}}$$

$$h_{n-1}(\gamma_{n-1} - \gamma_{n-2}) = h_{n-2}(\gamma_n - \gamma_{n-1})$$

$$h_{n-1}(\gamma_{n-1} - \gamma_{n-2}) - h_{n-2}(\gamma_n - \gamma_{n-1}) = 0$$

$$-h_{n-1}\gamma_{n-2} + (h_{n-2} + h_{n-1})\gamma_{n-1} - h_{n-2}\gamma_n = 0$$

de manera que el sistema $Ax = b$ es de la siguiente forma

$$A = \begin{pmatrix} -h_1 & h_0+h_1 & -h_0 & 0 & \cdots & \cdots & 0 \\ h_0 & 2(h_0+h_1) & h_1 & \ddots & & & \vdots \\ 0 & h_1 & 2(h_1+h_2) & h_2 & \ddots & & 0 \\ 0 & 0 & h_2 & 2(h_2+h_3) & h_3 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & & \\ & & & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} & 0 \\ 0 & \cdots & \cdots & 0 & -h_{n-1} & h_{n-1}+h_{n-2} & -h_{n-2} \end{pmatrix}$$

$$x = \begin{pmatrix} \gamma_0 \\ \gamma_1 \\ \gamma_2 \\ \\ \vdots \\ \\ \gamma_n \end{pmatrix}$$

$$b = \begin{pmatrix} 0 \\ \frac{3}{h_1}(\alpha_2 - \alpha_1) - \frac{3}{h_0}(\alpha_1 - \alpha_0) \\ \frac{3}{h_2}(\alpha_3 - \alpha_2) - \frac{3}{h_1}(\alpha_2 - \alpha_1) \\ \vdots \\ \frac{3}{h_{n-1}}(\alpha_n - \alpha_{n-1}) - \frac{3}{h_{n-2}}(\alpha_{n-1} - \alpha_{n-2}) \\ 0 \end{pmatrix}$$

2.2.3. Interpolación de Hermite Cúbico

La idea del interpolante Hermite Cúbico es construir polinomios de Hermite en cada subintervalo $[x_i, x_{i+1}]$ para $i = 0, 1, ..., n-1$.

Para obtener una expresión de cada polinomio f_i, desarrollemos a partir de la definición formal del polinomio de Hermite.

$$f_i(x) = \sum_{j=i}^{i+1} f(x_j)[1 - 2(x - x_j)L'_{i+1,j}(x_j)]L^2_{i+1,j}(x) + \sum_{j=i}^{i+1} f'(x_j)(x - x_j)L^2_{i+1,j}(x)$$

en este caso

$$L_{i+1,j}(x) = \prod_{\substack{k=i \\ k \neq j}}^{i+1} \frac{x - x_k}{x_j - x_k}$$

para $i = 0, 1, ..., n-1$.

De $f_i(x)$ obtenemos

$$f_i(x) = f(x_i)\left(1 - \frac{2(x - x_i)}{x_i - x_{i+1}}\right)\left(\frac{x - x_{i+1}}{x_i - x_{i+1}}\right)^2 + \cdots$$

$$+ f(x_{i+1})\left(1 - \frac{2(x - x_{i+1})}{x_{i+1} - x_i}\right)\left(\frac{x - x_i}{x_{i+1} - x_i}\right)^2 + \cdots$$

$$+ f'(x_i)(x - x_i)\left(\frac{x - x_{i+1}}{x_i - x_{i+1}}\right)^2 + f'(x_{i+1})(x - x_{i+1})\left(\frac{x - x_i}{x_{i+1} - x_i}\right)^2$$

sustituyendo en la ecuación

$$x_{i+1} - x_i = h_i,\ x - x_i = t,\ x - x_{i+1} = t - h_i$$

obtenemos

$$f_i(x) = f(x_i)\left(1+\frac{2t}{h_i}\right)\left(\frac{t-h_i}{h_i}\right)^2 + f(x_{i+1})\left(1-\frac{2(t-h_i)}{h_i}\right)\left(\frac{t}{h_i}\right)^2 + \cdots$$

$$+f'(x_i)t\left(\frac{t-h_i}{h_i}\right)^2 + f'(x_{i+1})(t-h_i)\left(\frac{t}{h_i}\right)^2$$

la nueva igualdad da como resultado

$$f_i(x) =$$

$$f(x_i) + f'(x_i) + \frac{3f(x_{i+1})t^2 + 3f(x_i)t^2 - f'(x_{i+1})t^2h_i - 2f'(x_i)t^2h_i}{h_i^2} + \cdots$$

$$+\frac{\frac{2f(x_i)t^3}{h_i} - \frac{2f(x_{i+1})t^3}{h_i} + f'(x_{i+1})t^3 + f'(x_i)t^3}{h_i^2} = \cdots$$

$$= f(x_i) + f'(x_i)t + \frac{3s_i - f'(x_{i+1}) - 2f'(x_i)}{h_i}t^2 + -\frac{2s_i - f'(x_{i+1}) - f'(x_i)}{h_i^2}t^3$$

definiendo

$$\alpha_i = f(x_i),\ \beta_i = f'(x_i),\ \gamma_i = \frac{3s_i - f'(x_{i+1}) - 2f'(x_i)}{h_i},\ \delta_i = -\frac{2s_i - f'(x_{i+1}) - f'(x_i)}{h_i^2}$$

obtenemos la expresión

$$f_i(x) = \alpha_i + \beta_i(x-x_i) + \gamma_i(x-x_i)^2 + \delta_i(x-x_i)^3$$

que es el polinomio de Hermite Cúbico.

Notemos la necesidad de obtener la derivada para calcular cada uno de los polinomios, sin embargo no siempre se cuenta con la función f para aproximar f' ni tampoco con los valores de $f'(x_i)$ para $i = 0, 1, ..., n-1$, por lo cual se han desarrollado métodos para aproximar $f'(x_i)$ usando únicamente los puntos (x_i, y_i).

Interpolación Cúbica de Akima

Para comenzar con el desarrollo del interpolante de Akima, es necesario partir de las siguientes construcciones geométricas

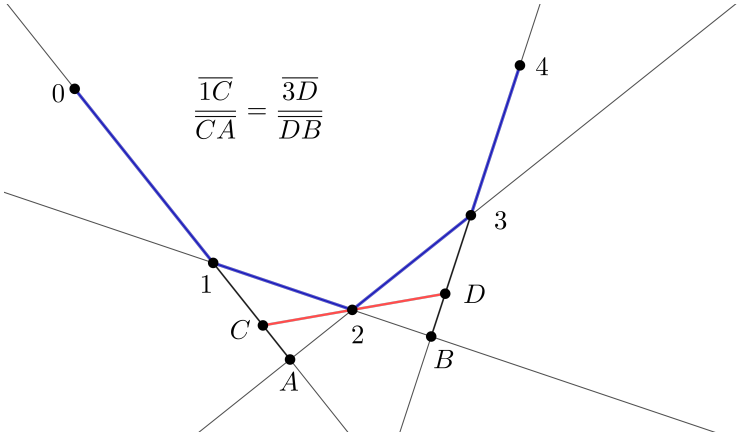


Figura 2.10

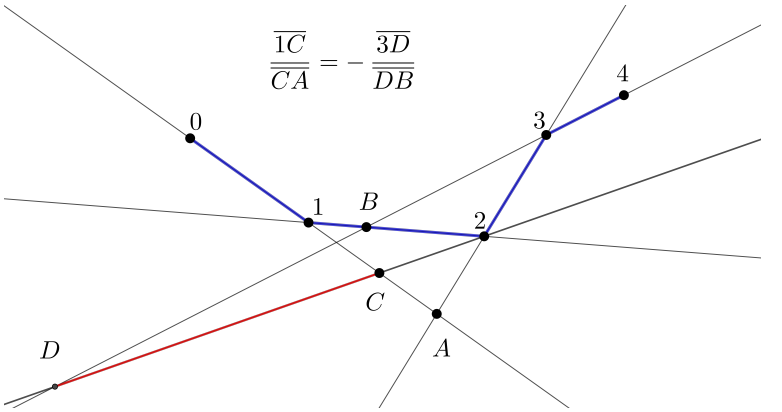


Figura 2.11

El interpolante aproxima la derivada en (x_i, y_i) para $i = 0, 1, ..., n$, con el propósito de tener información más completa del comportamiento de la pendiente en cada par ordenado, se contemplan dos puntos anteriores y posteriores al que se desea aproximar, en este caso aproximaremos f' en el punto 2 de las figuras 2.10 y 2.11. En las construcciones geométricas anteriores, A es la intersección de la recta formada por 0 y 1 con la que pasa por los puntos 2 y 3 mientras que B es la intersección de la recta formada por 1 y 2 y la que pasa por 4 y 3, C y D corresponde a la intersección de la recta tangente que pasa por 2 con las rectas que pasan por 0 y 1 y la recta que pasa por 4 y 3 respectivamente, el problema ahora consiste en aproximar la pendiente de la recta tangente a 2 en términos de las pendientes de las rectas formadas por los puntos anteriores y posteriores a 2, con lo cual aproximaremos f' en 2.

Un supuesto más que se requiere para calcular la pendiente del segmento $\overline{CD}$ en la figura 2.10

$$\frac{\overline{1C}}{\overline{CA}} = \frac{\overline{3D}}{\overline{DB}} \tag{2.8}$$

de la misma forma para 2.11

$$\frac{\overline{1C}}{\overline{CA}} = \frac{\overline{3D}}{\overline{BD}} \tag{2.9}$$

geométricamente la distancia del segmento $\overline{DB}$ es igual a la de $\overline{BD}$ pero en sentido inverso, es decir que $\overline{DB} = -\overline{BD}$, con lo cual podemos reescribir la ecuación 2.9 como sigue

$$\frac{\overline{1C}}{\overline{CA}} = -\frac{\overline{3D}}{\overline{DB}} \tag{2.10}$$

usando 2.8 y 2.10 llegamos a la siguiente igualdad

$$\left|\frac{\overline{1C}}{\overline{CA}}\right| = \left|\frac{\overline{3D}}{\overline{DB}}\right| \tag{2.11}$$

denotemos las coordenadas de los puntos 0,1,2,3,4,A,B,C y D con (x, y) y los subíndices 0,1,2,3,4,A,B,C y D para cada punto respectivamente, también definimos

$$a_i = x_{i+1} - x_i$$

$$b_i = y_{i+1} - y_i$$

para $i = 0, 1, 3, 4$.

Para encontrar f' en el punto 2 definiremos a $f'(2)$ como

$$f'(2) = \frac{y_d - y_c}{x_d - x_c}$$

que es la pendiente del segmento de recta $\overline{CD}$ que pasa por el punto 2. Con las figuras 2.10 y 2.11 podemos obtener las siguientes igualdades

$$\frac{y_a - y_1}{x_a - x_1} = \frac{y_c - y_1}{x_c - x_1} = \frac{b_0}{a_0} \tag{2.12}$$

$$\frac{y_3 - y_b}{x_3 - x_b} = \frac{y_3 - y_d}{x_3 - x_d} = \frac{b_3}{a_3} \tag{2.13}$$

$$\frac{y_2 - y_a}{x_2 - x_a} = \frac{b_2}{a_2} \tag{2.14}$$

$$\frac{y_b - y_2}{x_b - x_2} = \frac{b_1}{a_1} \tag{2.15}$$

$$\frac{y_2 - y_c}{x_2 - x_c} = \frac{y_d - y_2}{x_d - x_2} = f'(2) \tag{2.16}$$

usando la ecuación 2.12 obtenemos

$$\frac{y_a - y_1 + (y_2 - y_2)}{x_a - x_1 + (x_2 - x_2)} = \frac{y_a - y_2 + y_2 - y_1}{x_a - x_2 + x_2 - x_1} = \frac{y_2 - y_a - b_1}{x_2 - x_a - a_1} = \frac{b_0}{a_0} \tag{2.17}$$

de la igualdad 2.17 se tiene

$$x_2 - x_a = \frac{a_0(y_2 - y_a - b_1)}{b_0} + a_1 = \frac{a_0(y_2 - y_a - b_1) + b_0 a_1}{b_0}$$

con 2.14

$$x_2 - x_a = \frac{a_0 \dfrac{b_2}{a_2}(x_2 - x_a) - a_0 b_1 + b_0 a_1}{b_0} = \frac{a_0 b_2 (x_2 - x_a) - a_2 a_0 b_1 + a_2 b_0 a_1}{a_2 b_0}$$

$$\frac{a_2 b_0 (x_2 - x_a) - a_0 b_2 (x_2 - x_a)}{a_2} = -a_0 b_1 + a_1 b_0$$

$$\frac{x_2 - x_a}{a_2} = \frac{a_0 b_1 - a_1 b_0}{a_0 b_2 - a_2 b_0} \tag{2.18}$$

de las ecuaciones 2.13 y 2.15

$$\frac{y_3 - y_b + (y_2 - y_2)}{x_3 - x_b + (x_2 - x_2)} = \frac{-y_b + y_2 + y_2 - y_1}{-x_b + x_2 + x_3 - x_2} = \frac{y_b - y_2 - b_2}{x_b - x_2 - a_2} = \frac{b_3}{a_3}$$

$$x_b - x_2 = \frac{a_3(y_b - y_2 - b_2)}{b_3} + a_2 = \frac{a_3(y_b - y_2 - b_2) + b_3 a_2}{b_3}$$

$$x_b - x_2 = \frac{a_3 \dfrac{b_1}{a_1}(x_b - x_2) - a_3 b_2 + b_3 a_2}{b_3} = \frac{a_3 b_1 (x_b - x_2) - a_1 a_3 b_2 + a_1 b_3 a_2}{a_1 b_3}$$

$$\frac{a_1 b_3 (x_b - x_2) - a_3 b_1 (x_b - x_2)}{a_1} = -a_3 b_2 + a_2 b_3$$

$$\frac{x_b - x_2}{a_1} = \frac{a_2 b_3 - a_3 b_2}{a_1 b_3 - a_3 b_1} \tag{2.19}$$

de 2.12 y 2.16

$$\frac{y_c - y_1 + (y_2 - y_2)}{x_c - x_1 + (x_2 - x_2)} = \frac{y_c - y_2 + y_2 - y_1}{x_c - x_2 + x_2 - x_1} = \frac{y_2 - y_c - b_1}{x_2 - x_c - a_1} = \frac{b_0}{a_0}$$

$$x_2 - x_c = \frac{a_0(y_2 - y_c - b_1)}{b_0} + a_1 = \frac{a_0(y_2 - y_c - b_1) + b_0 a_1}{b_0}$$

$$x_2 - x_c = \frac{a_0 f'(2)(x_2 - x_c) - a_0 b_1 + b_0 a_1}{b_0}$$

$$b_0(x_2 - x_c) - a_0 f'(2)(x_2 - x_c) = -a_0 b_1 + a_1 b_0$$

$$x_2 - x_c = \frac{a_0 b_1 - a_1 b_0}{a_0 f'(2) - b_0} \tag{2.20}$$

de manera similar utilizamos 2.13 y 2.16

$$\frac{y_3 - y_d + (y_2 - y_2)}{x_3 - x_d + (x_2 - x_2)} = \frac{-y_d + y_2 - y_2 + y_3}{-x_d - x_2 + x_2 + x_3} = \frac{y_d - y_2 - b_2}{x_d - x_2 - a_2} = \frac{b_3}{a_3}$$

$$x_d - x_2 = \frac{a_3(y_d - y_2 - b_2)}{b_3} + a_2 = \frac{a_3(y_d - y_2 - b_2) + b_3 a_2}{b_3}$$

$$x_d - x_2 = \frac{a_3 f'(2)(x_d - x_2) - a_3 b_2 + b_3 a_2}{b_3}$$

$$b_3(x_d - x_2) - a_3 f'(2)(x_d - x_2) = -a_3 b_2 + a_2 b_3$$

$$x_d - x_2 = \frac{a_2 b_3 - a_3 b_2}{b_3 - a_3 f'(2)} \tag{2.21}$$

nuevamente con el supuesto de 2.11 es posible ver a la ecuación como sigue

$$\left|\frac{x_1 - x_c}{x_c - x_a}\right| = \left|\frac{x_3 - x_d}{x_d - x_b}\right|$$

$$\left|\frac{x_1 - x_c + (x_2 - x_2)}{x_c - x_a + (x_2 - x_2)}\right| = \left|\frac{x_3 - x_d + (x_2 - x_2)}{x_d - x_b + (x_2 - x_2)}\right|$$

$$\left|\frac{x_2 - x_c - (x_2 - x_1)}{x_2 - x_a - (x_2 - x_c)}\right| = \left|\frac{x_3 - x_2 - (x_d - x_2)}{x_d - x_2 - (x_b - x_2)}\right|$$

$$\left|\frac{(x_2 - x_c) - a_1}{(x_2 - x_a) - (x_2 - x_c)}\right| = \left|\frac{a_2 - (x_d - x_2)}{(x_d - x_2) - (x_b - x_2)}\right| \tag{2.22}$$

sustituyendo en 2.22 las ecuaciones 2.18,2.19,2.20 y 2.21 obtenemos

$$\left|\frac{\frac{a_0b_1-a_1b_0}{a_0f'(2)-b_0}-a_1}{a_2\frac{a_0b_1-a_1b_0}{a_0b_2-a_2b_0}-\frac{a_0b_1-a_1b_0}{a_0f'(2)-b_0}}\right| = \left|\frac{a_2-\frac{a_2b_3-a_3b_2}{b_3-a_3f'(2)}}{\frac{a_2b_3-a_3b_2}{b_3-a_3f'(2)}-a_1\frac{a_2b_3-a_3b_2}{a_1b_3-a_3b_1}}\right|$$

$$\left|-\frac{(a_0b_2-a_2b_0)\,(f'(2)a_1-b_1)}{(a_0b_1-a_1b_0)\,(f'(2)a_2-b_2)}\right| = \left|-\frac{(a_1b_3-a_3b_1)\,(f'(2)a_2-b_2)}{(a_2b_3-a_3b_2)\,(f'(2)a_1-b_1)}\right|$$

$$\begin{gathered}|(a_0b_2-a_2b_0)\,(f'(2)a_1-b_1)(a_2b_3-a_3b_2)\,(f'(2)a_1-b_1)| = \\ |\,(a_1b_3-a_3b_1)\,(f'(2)a_2-b_2)\,(a_0b_1-a_1b_0)\,(f'(2)a_2-b_2)|\end{gathered}$$

$$|(a_0b_2-a_2b_0)(a_2b_3-a_3b_2)\,|\,(f'(2)a_1-b_1)^2 =$$

$$|\,(a_1b_3-a_3b_1)\,(a_0b_1-a_1b_0)\,|\,(f'(2)a_2-b_2)^2 \tag{2.23}$$

usando las figuras 2.10 y 2.11 podemos llegar a las siguientes desigualdades

$$0 < \frac{b_2}{a_2}+\frac{b_1}{a_1} = \frac{a_1b_2+a_2b_1}{a_1a_2}$$

$$\frac{b_1b_2}{a_1a_2} \le 0$$

$$f'(2)^2a_1a_2 \le 0$$

$$\frac{f'(2)^2a_1a_2}{a_1a_2}-f'(2)\frac{a_1b_2+a_2b_1}{a_1a_2}+\frac{b_1b_2}{a_1a_2} \le 0$$

$$\frac{f'(2)^2a_1a_2+(-a_1b_2-b_1a_2)\,f'(2)+b_1b_2}{a_1a_2} \le 0$$

$$\frac{(f'(2)a_1-b_1)(f'(2)a_2-b_2)}{a_1a_2} \le 0$$

$$(f'(2)a_1-b_1)(f'(2)a_2-b_2) \le 0 \tag{2.24}$$

con las ecuaciones 2.23 y 2.24 obtenemos

$$\begin{gathered}|(a_0b_2-a_2b_0)(a_2b_3-a_3b_2)\,|^{\frac{1}{2}}|f'(2)a_1-b_1| = \\ |\,(a_1b_3-a_3b_1)\,(a_0b_1-a_1b_0)\,|^{\frac{1}{2}}|f'(2)a_2-b_2|\end{gathered}$$

$$\begin{gathered}|(a_0b_2-a_2b_0)(a_2b_3-a_3b_2)\,|^{\frac{1}{2}}(f'(2)a_1-b_1) = \\ |\,(a_1b_3-a_3b_1)\,(a_0b_1-a_1b_0)\,|^{\frac{1}{2}}(b_2-f'(2)a_2)\end{gathered}$$

$$f(2)' = \frac{|\,(a_1b_3 - a_3b_1)\,(a_0b_1 - a_1b_0)\,|^{\frac{1}{2}}b_2 + |(a_0b_2 - a_2b_0)(a_2b_3 - a_3b_2)\,|^{\frac{1}{2}}b_1}{|\,(a_1b_3 - a_3b_1)\,(a_0b_1 - a_1b_0)\,|^{\frac{1}{2}}a_2 + |(a_0b_2 - a_2b_0)(a_2b_3 - a_3b_2)\,|^{\frac{1}{2}}a_1} \tag{2.25}$$

definamos la siguiente notación

$$R_{i,j} = a_ib_j - a_jb_i$$

con lo cual reescribimos la ecuación 2.25 como

$$f(2)' = \frac{|R_{1,3}R_{0,1}|^{\frac{1}{2}}b_2 + |R_{0,2}R_{2,3}|^{\frac{1}{2}}b_1}{|R_{1,3}R_{0,1}|^{\frac{1}{2}}b_2 + |R_{0,2}R_{2,3}|^{\frac{1}{2}}b_1}$$

$$= \frac{\left|\frac{R_{0,2}R_{2,3}}{{a_2}^2a_0a_3}\right|^{\frac{1}{2}}|{a_2}^2a_0a_3|^{\frac{1}{2}}\,b_1 + \left|\frac{R_{0,1}R_{1,3}}{{a_1}^2a_0a_3}\right|^{\frac{1}{2}}\left|{a_1}^2a_0a_3\right|^{\frac{1}{2}}\,b_2}{\left|\frac{R_{0,2}R_{2,3}}{{a_2}^2a_0a_3}\right|^{\frac{1}{2}}|{a_2}^2a_0a_3|^{\frac{1}{2}}\,a_1 + \left|\frac{R_{0,1}R_{1,3}}{{a_1}^2a_0a_3}\right|^{\frac{1}{2}}\left|{a_1}^2a_0a_3\right|^{\frac{1}{2}}\,a_2}$$

$$= \frac{\left|\left(\frac{b_2}{a_2} - \frac{b_0}{a_0}\right)\left(\frac{b_3}{a_3} - \frac{b_2}{a_2}\right)\right|^{\frac{1}{2}}|a_2|b_1 + \left|\left(\frac{b_1}{a_1} - \frac{b_0}{a_0}\right)\left(\frac{b_3}{a_3} - \frac{b_1}{a_1}\right)\right|^{\frac{1}{2}}|a_1|b_2}{\left|\left(\frac{b_2}{a_2} - \frac{b_0}{a_0}\right)\left(\frac{b_3}{a_3} - \frac{b_2}{a_2}\right)\right|^{\frac{1}{2}}|a_2|a_1 + \left|\left(\frac{b_1}{a_1} - \frac{b_0}{a_0}\right)\left(\frac{b_3}{a_3} - \frac{b_1}{a_1}\right)\right|^{\frac{1}{2}}|a_1|a_2}$$

definimos $|a_2|$ y $|a_1|$ como sigue:

$$|a_1| = (\text{signo de } a_1)a_1$$

$$|a_2| = (\text{signo de } a_2)a_2$$

y también definimos

$$p_1 = (\text{signo de } a_2)|(s_2 - s_0)(s_3 - s_2)|^{\frac{1}{2}}$$

$$p_2 = (\text{signo de } a_1)|(s_1 - s_0)(s_3 - s_1)|^{\frac{1}{2}}$$

con lo cual $f'(2)$ puede reescribirse como:

$$f'(2) = \frac{p_1a_2b_1 + p_2a_1b_2}{p_1a_2a_1 + p_2a_1a_2} = \frac{p_1\dfrac{b_1}{a_1} + p_2\dfrac{b_2}{a_2}}{p_1 + p_2} = \frac{p_1s_1 + p_2s_2}{p_1 + p_2} \tag{2.26}$$

podemos observar que la aproximación a $f'(2)$ depende de las pendientes anteriores y posteriores al punto 2, es decir s_0, s_1, s_2 y s_3 y además cumple las siguientes propiedades:

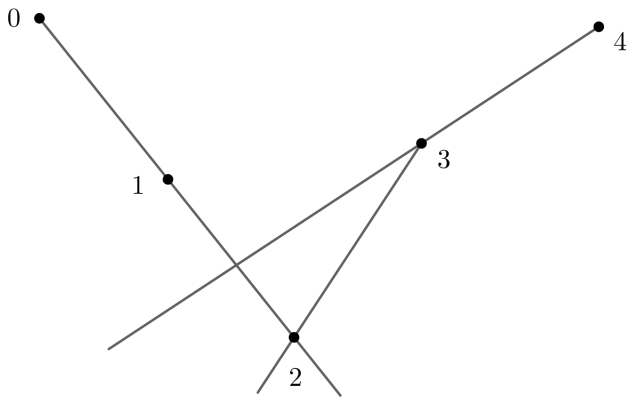


Figura 2.12: Ejemplo cuando se cumple $s_0 = s_1, s_0 \neq s_2$ y $s_3 \neq s_2$.

1. Si $s_0 = s_1, s_0 \neq s_2$ y $s_3 \neq s_2$ entonces $f'(2) = s_0 = s_1$.
2. Si $s_2 = s_3, s_0 \neq s_1$ y $s_3 \neq s_1$ entonces $f'(2) = s_1 = s_2$.

Las propiedades 1 y 2 son muy importantes ya que como vemos en las figuras 2.12 y 2.13 en ocasiones, por la posición de los puntos no podemos encontrar las intersecciones A, B ,C y D, aun así la expresión 2.26 ofrece una aproximación, sin embargo dicha expresión no está definida cuando $s_0 = s_1 = s_2$

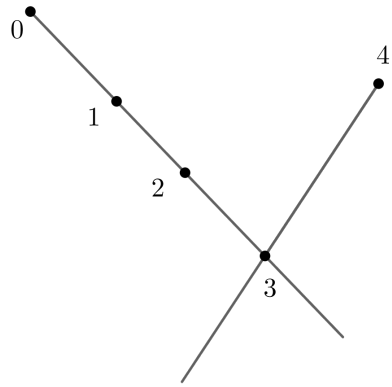


Figura 2.14: Condición $s_0 = s_1 = s_2$.

de manera intuitiva no es difícil dar el valor de $f'(2) = s_0 = s_1 = s_2$, otro inconveniente con la expresión 2.26 surge cuando $s_1 = s_3, s_2 \neq s_0$ y $s_3 \neq s_2$ ya que $f'(2) = s_1$ y de forma similar $f'(2) = s_2$ cuando $s_2 = s_0, s_1 \neq s_0$ y $s_3 \neq s_1$, lo cual en ambos caso no es una buena aproximación a $f'(2)$ como podemos ver en las siguientes figuras:

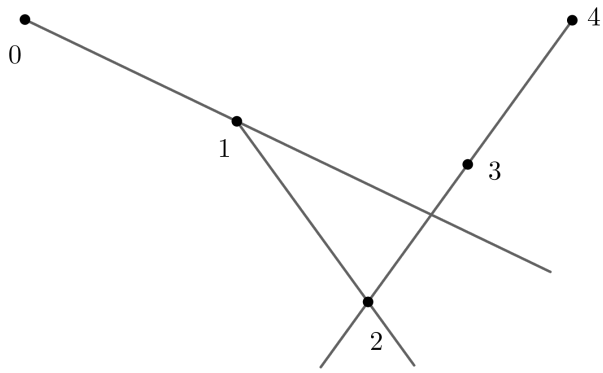


Figura 2.13: Ejemplo cuando se cumple $s_2 = s_3, s_0 \neq s_1$ y $s_3 \neq s_1$.

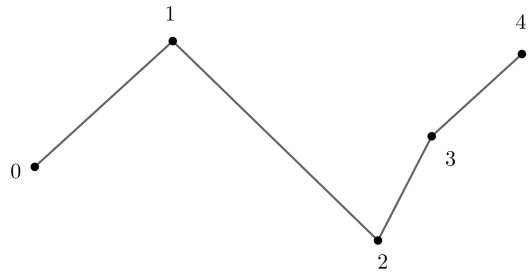


Figura 2.15

debido a los inconvenientes mencionados, es necesario modificar p_1 y p_2 de tal manera que las propiedades 1 y 2 se conserven, basado en las propiedades 1 y 2, p_1 y p_2 se reescriben como sigue:

$$p_1 = |s_3 - s_2|$$

$$p_2 = |s_1 - s_0|$$

[3, pág. 600] con lo cual $f'(2)$ se define como

$$f'(2) = \frac{|s_3 - s_2|s_1 + |s_1 - s_0|s_2}{|s_3 - s_2| + |s_1 - s_0|} \tag{2.27}$$

notemos que la ecuación 2.27 no está definida sólo cuando $s_0 = s_1$ y $s_2 = s_3$, que se puede observar en la siguiente figura

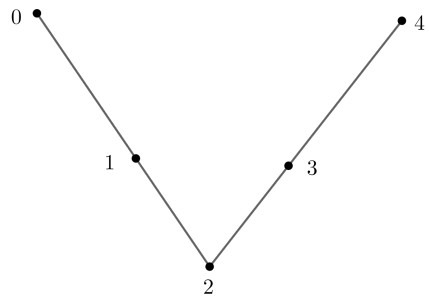


Figura 2.16

en este caso, una aproximación razonable a $f'(2)$ es el promedio de las pendientes s_1 y s_2.

$$f'(2) = \frac{s_1 + s_2}{2} \tag{2.28}$$

Con lo obtenido anteriormente, podemos generalizar el concepto para n puntos, entonces la expresión final de la interpolación cúbica de Akima es de la siguiente forma

$$f'(x_i) = \begin{cases} \dfrac{s_{i-1} + s_i}{2} & si \quad s_{i+1} = s_i \ y \ s_{i-1} = s_{i-2} \\ \dfrac{|s_{i+1} - s_i| s_{i-1} + |s_{i-1} - s_{i-2}| s_i}{|s_{i+1} - s_i| + |s_{i-1} - s_{i-2}|} & en \ otro \ caso \end{cases}$$

para $i = 0, 1, 2, ..., n$.

Debemos observar, que para el cálculo de $f'(x_i)$ necesitamos 2 puntos extra en cada punto frontera (x_0, y_0) y (x_n, y_n), donde (x_{-2}, y_{-2}),(x_{-1}, y_{-1}) son puntos anteriores a (x_0, y_0) y (x_{n+1}, y_{n+1}),(x_{n+2}, y_{n+2}) puntos posteriores a (x_n, y_n), además dichos puntos cumplen

$$\boldsymbol{x_{-2}} < \boldsymbol{x_{-1}} < x_0 < x_1 < \cdots < x_{n-1} < x_n < \boldsymbol{x_{n+1}} < \boldsymbol{x_{n+2}}$$

para calcular los nuevos puntos

$$(x_{n-2}, y_{n-2}), (x_{n-1}, y_{n-1}), (x_n, y_n), (x_{n+1}, y_{n+1}), (x_{n+2}, y_{n+2})$$

debemos asumir lo siguiente

$$d_1 = x_{n+2} - x_n = x_{n+1} - x_{n-1} = x_n - x_{n-2} \tag{2.29}$$

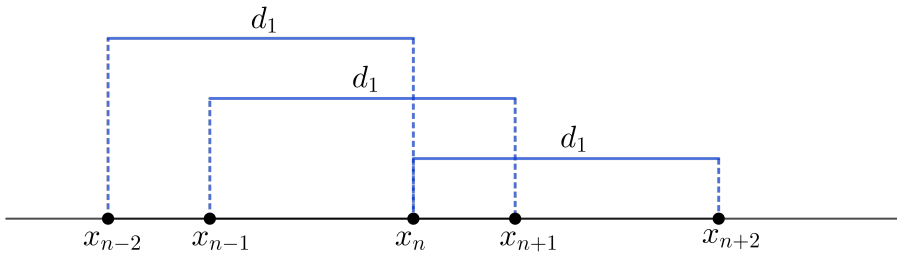


Figura 2.17

y también que (x_i, y_i) para $i = n-2, n-1, ..., n+2$ pertenecen a una parábola

$$a_0 + a_1(x - x_n) + a_2(x - x_n)^2 \tag{2.30}$$

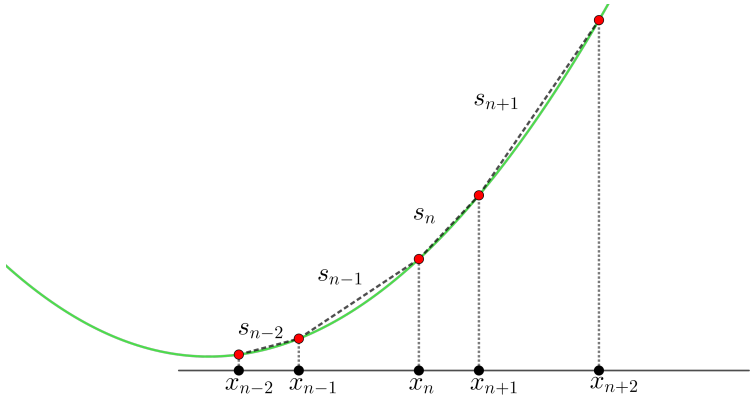


Figura 2.18

de manera análoga en los puntos

$$(x_{-2}, y_{-2}), (x_{-1}, y_{-1}), (x_0, y_0), (x_1, y_1), (x_2, y_2)$$

suponemos que

$$d_2 = x_2 - x_0 = x_1 - x_{-1} = x_0 - x_{-2} \tag{2.31}$$

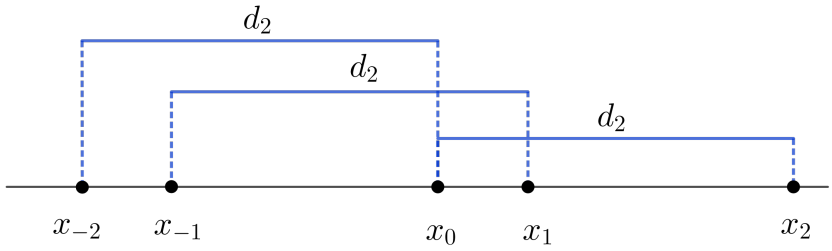


Figura 2.19

y que los puntos (x_i, y_i) para $i = -2, -1, ..., 2$ forman parte de la parábola

$$g_0 + g_1(x - x_0) + g_2(x - x_0)^2 \tag{2.32}$$

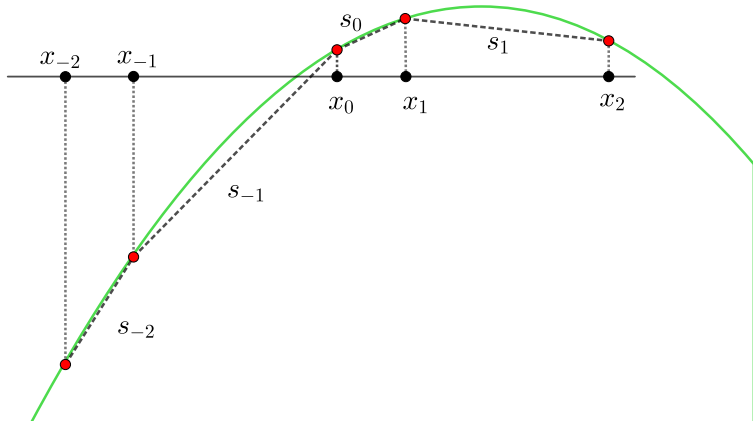


Figura 2.20

gracias a las ecuaciones anteriores podemos verificar lo siguiente

$$s_{n+1} - s_n = s_{n+1} - s_{n-1} = s_{n-1} - s_{n-2}$$

$$s_1 - s_0 = s_0 - s_{-1} = s_{-1} - s_{-2}$$

primero obtenemos

$$y_{n+i} = a_0 + a_1(x_{n+i} - x_n) + a_2(x_{n+i} - x_n)^2$$

$$y_i = g_0 + g_1(x_i - x_0) + g_2(x_i - x_0)^2$$

para $i = -2, -1, 0, 1, 2$, desarrollando las diferencias obtenemos

$$\frac{y_{n+2} - y_{n+1}}{x_{n+2} - x_{n+1}} - \frac{y_{n+1} - y_n}{x_{n+1} - x_n} =$$

$$\frac{(a_0 + a_1(x_{n+2} - x_n) + a_2(x_{n+2} - x_n)^2) - (a_0 + a_1(x_{n+1} - x_n) + a_2(x_{n+1} - x_n)^2)}{x_{n+2} - x_{n+1}} -$$

$$\frac{a_0 + a_1(x_{n+1} - x_n) + a_2(x_{n+1} - x_n)^2 - a_0}{x_{n+1} - x_n} = a_2(x_{n+2} - x_n)$$

$$\frac{y_{n+1} - y_n}{x_{n+1} - x_n} - \frac{y_n - y_{n-1}}{x_n - x_{n-1}} =$$

$$\frac{a_0 + a_1(x_{n+1} - x_n) + a_2(x_{n+1} - x_n)^2 - a_0}{x_{n+1} - x_n} -$$

$$\frac{a_0 - (a_0 + a_1(x_{n-1} - x_n) + a_2(x_{n-1} - x_n)^2)}{x_n - x_{n-1}} = a_2(x_{n+1} - x_{n-1})$$

$$\frac{y_n - y_{n-1}}{x_n - x_{n-1}} - \frac{y_{n-1} - y_{n-2}}{x_{n-1} - x_{n-2}} =$$

$$\frac{a_0 - (a_0 + a_1(x_{n-1} - x_n) + a_2(x_{n-1} - x_n)^2)}{x_n - x_{n-1}} -$$

$$\frac{(a_0 + a_1(x_{n-1} - x_n) + a_2(x_{n-1} - x_n)^2) - (a_0 + a_1(x_{n-2} - x_n) + a_2(x_{n-2} - x_n)^2)}{x_{n-1} - x_{n-2}}$$

$$= a_2(x_n - x_{n-2})$$

usando la ecuación 2.29

$$a_2(x_{n+2} - x_n) = a_2(x_{n+1} - x_{n-1}) = a_2(x_n - x_{n-2})$$

por lo tanto

$$\frac{y_{n+2} - y_{n+1}}{x_{n+2} - x_{n+1}} - \frac{y_{n+1} - y_n}{x_{n+1} - x_n} = \frac{y_{n+1} - y_n}{x_{n+1} - x_n} - \frac{y_n - y_{n-1}}{x_n - x_{n-1}} = \frac{y_n - y_{n-1}}{x_n - x_{n-1}} - \frac{y_{n-1} - y_{n-2}}{x_{n-1} - x_{n-2}} \tag{2.33}$$

de manera similar calculamos

$$\frac{y_2 - y_1}{x_2 - x_1} - \frac{y_1 - y_0}{x_1 - x_0} =$$

$$\frac{(g_0 + g_1(x_2 - x_0) + g_2(x_2 - x_0)^2) - (g_0 + g_1(x_1 - x_0) + g_2(x_1 - x_0)^2)}{x_2 - x_1} -$$

$$\frac{g_0 + g_1(x_1 - x_0) + g_2(x_1 - x_0)^2 - g_0}{x_1 - x_0} = g_2(x_2 - x_0)$$

$$\frac{y_1 - y_0}{x_1 - x_0} - \frac{y_0 - y_{-1}}{x_0 - x_{-1}} =$$

$$\frac{g_0 + g_1(x_1 - x_0) + g_2(x_1 - x_0)^2 - g_0}{x_1 - x_0} -$$

$$\frac{g_0 - (g_0 + g_1(x_{-1} - x_0) + g_2(x_{-1} - x_0)^2)}{x_0 - x_{-1}} = g_2(x_1 - x_{-1})$$

$$\frac{y_0 - y_{-1}}{x_0 - x_{-1}} - \frac{y_{-1} - y_{-2}}{x_{-1} - x_{-2}} =$$

$$\frac{g_0 - (g_0 + g_1(x_{-1} - x_0) + g_2(x_{-1} - x_0)^2)}{x_0 - x_{-1}} -$$

$$\frac{(g_0 + g_1(x_{-1} - x_0) + g_2(x_{-1} - x_0)^2) - (g_0 + g_1(x_{-2} - x_0) + g_2(x_{-2} - x_0)^2)}{x_{-1} - x_{-2}}$$

$$= g_2(x_0 - x_{-2})$$

y con la ecuación 2.31 se tiene

$$g_2(x_2 - x_0) = g_2(x_1 - x_{-1}) = g_2(x_0 - x_{-2})$$

por lo tanto

$$\frac{y_2 - y_1}{x_2 - x_1} - \frac{y_1 - y_0}{x_1 - x_0} = \frac{y_1 - y_0}{x_1 - x_0} - \frac{y_0 - y_{-1}}{x_0 - x_{-1}} = \frac{y_0 - y_{-1}}{x_0 - x_{-1}} - \frac{y_{-1} - y_{-2}}{x_{-1} - x_{-2}} \quad (2.34)$$

usando las ecuaciones 2.29, 2.31, 2.33 y 2.34 podemos obtener el valor los puntos extras necesarios,

$$x_{n+2} = 2x_n - x_{n-2}$$

$$x_{n+1} = x_n + x_{n-1} - x_{n-2}$$

$$y_{n+1} = \left(\frac{2(y_n - y_{n-1})}{x_n - x_{n-1}} - \frac{y_{n-1} - y_{n-2}}{x_{n-1} - x_{n-2}}\right)(x_{n+1} - x_n) + y_n$$

$$y_{n+2} = \left(\frac{2(y_{n+1} - y_n)}{x_{n+1} - x_n} - \frac{y_n - y_{n-1}}{x_n - x_{n-1}}\right)(x_{n+2} - x_{n+1}) + y_{n+1}$$

$$x_{-2} = 2x_0 - x_2$$

$$x_{-1} = x_0 + x_1 - x_2$$

$$y_{-1} = y_0 - \left(\frac{2(y_1 - y_0)}{x_1 - x_0} - \frac{y_2 - y_1}{x_2 - x_1}\right)(x_0 - x_{-1})$$

$$y_{-2} = y_{-1} - \left(\frac{2(y_0 - y_{-1})}{x_0 - x_{-1}} - \frac{y_1 - y_0}{x_1 - x_0}\right)(x_{-1} - x_{-2})$$

calculando estos puntos tenemos todo lo necesario para implementar el interpolante.

Interpolación Restringida

A diferencia del método anterior, la interpolación restringida no requiere de puntos extra. Usando como base la "media armónica" para dos números a_1 y a_2:

$$\frac{2a_1a_2}{a_1 + a_2} = \frac{2}{\frac{1}{a_1} + \frac{1}{a_2}}$$

se usa lo anterior para hallar una aproximación a f' como se muestra en la figura siguiente

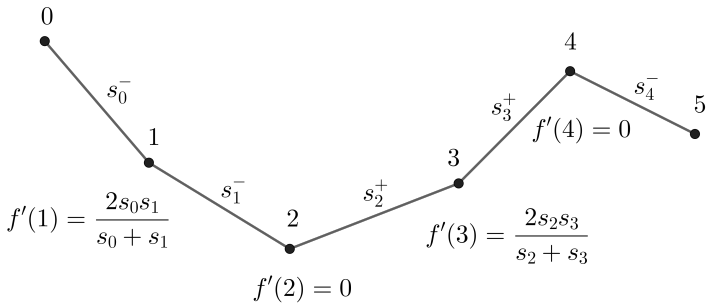


Figura 2.21: Uso de la media armónica para aproximar f' .

podemos observar que para aproximar f' en los puntos 1 y 3, se calculan las pendientes con el punto anterior y posterior de 1 y 3 respectivamente, después se calcula la media armónica de ambas pendientes, para los puntos 2 y 4 vemos que en las pendientes alrededor de cada punto son de signo distinto, en este caso se asume que hay un mínimo local y un máximo local en 2 y 4 respectivamente y se da por hecho que $f' = 0$, con este razonamiento podemos generalizar la aproximación a f' mediante la siguiente definición

$$f'(x_i) = \begin{cases} 0 & \textit{si la pendiente cambia de signo} \\ \dfrac{2s_{i-1}s_i}{s_{i-1}+s_i} = \dfrac{2}{1/s_{i-1}+1/s_i} & \textit{en otro caso} \end{cases}$$

para $i = 1, 2, ..., n-1$.

Es importante notar que la aproximación a f' no estará definida si $s_{i-1} = s_i = 0$, para obtener la aproximación en los puntos frontera, usamos las siguientes condiciones

$$f''(x_0) = f''(x_n) = 0$$

$$f''(x_0) = f_0''(x_0) = 2\gamma_0 + 6\delta_0(x_0 - x_0) = 0$$

$$2\gamma_0 = 0$$

sustituyendo y despejando

$$2\frac{3s_0 - f'(x_1) - 2f'(x_0)}{(x_1 - x_0)} = 0$$

$$2f'(x_0) = 3s_0 - f'(x_1)$$

$$\boxed{f'(x_0) = \frac{3}{2}s_0 - \frac{1}{2}f'(x_1)}$$

de forma análoga

$$f''(x_n) = f_{n-1}''(x_n) = 0$$

$$2\frac{3s_{n-1} - f'(x_n) - 2f'(x_{n-1})}{h_{n-1}} - 6\frac{2s_{n-1} - f'(x_n) - f'(x_{n-1})}{h_{n-1}^2}h_{n-1} = 0$$

$$3s_{n-1} - f'(x_n) - 2f'(x_{n-1}) - 6s_{n-1} + 3f'(x_n) + 3f'(x_{n-1}) = 0$$

$$-2f'(x_n) = -3s_{n-1} + f'(x_{n-1})$$

$$\boxed{f'(x_n) = \frac{3}{2}s_{n-1} - \frac{1}{2}f'(x_{n-1})}$$

Interpolación de Preservación Monótona

La propuesta para estimar f' en cada punto es la siguiente

$$f'(x_i) = \begin{cases} 0 & \text{si} \quad s_{i-1}s_i \leq 0 \\ \left(\dfrac{h_{i-1}+2h_i}{3(h_i+h_{i-1})s_{i-1}} + \dfrac{2h_{i-1}+h_i}{3(h_i+h_{i-1})s_i}\right)^{-1} & \text{si} \quad s_{i-1}s_i > 0 \end{cases}$$

para $i = 1, 2, ..., n-1$.

en los puntos frontera se define como

$$f'(x_0) = \begin{cases} 0 & \text{si} \quad g_0 s_0 \leq 0 \\ 3s_0 & \text{si} \quad s_0 s_1 \leq 0 \ y \ |g_0| > 3|s_0| \\ g_0 & \textit{en otro caso} \end{cases}$$

$$f'(x_n) = \begin{cases} 0 & \text{si} \quad g_n s_{n-1} \leq 0 \\ 3s_{n-1} & \text{si} \quad s_{n-1} s_{n-2} \leq 0 \ y \ |g_n| > 3|s_{n-1}| \\ g_n & \textit{en otro caso} \end{cases}$$

donde

$$g_0 = \frac{2h_0 + h_1}{h_0 + h_1} s_0 - \frac{h_0}{h_0 + h_1} s_1$$

$$g_n = \frac{2h_{n-1} + h_{n-2}}{h_{n-1} + h_{n-2}} s_{n-1} - \frac{h_{n-1}}{h_{n-1} + h_{n-2}} s_{n-2}$$

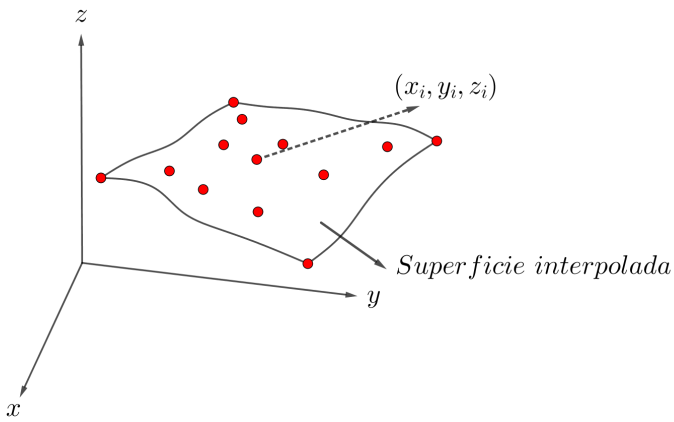
[1, pág. 7]

Hemos estudiado diferentes métodos de interpolación univaridada, comparando las características de cada uno, estos métodos servirán como base para construir un método bivariado de interpolación y poder determinar cual de ellos ofrece mejores resultados.

Capítulo 3

Interpolación Bivariada

En el capítulo anterior se estudiaron métodos de interpolación en una variable, es decir, dado un conjunto de puntos (x_i, y_i), para $i = 0, 1, \ldots, n$, obtenemos valores intermedios entre cada uno de ellos, en este capítulo, se analizarán métodos para interpolar en forma bivariada, donde dado un conjuntos de puntos (x_i, y_i, z_i) para $i = 1, 2, \ldots, n$, aproximaremos valores que están dentro de la región que forman estos puntos.



3.1. Interpolación Bivariada Mediante Interpolaciones Univariadas (IBMIU)

El primer acercamiento para interpolar en dos dimensiones es usar los interpolantes de una variable, ya que la idea principal es realizar una serie de interpolaciones primero sobre el eje (x, z) y finalmente una vez aproximados la serie de valores interpolar sobre el eje (y, z), es importante resaltar que la implementación del propuesto es posible si los puntos están regularmente distribuidos.

A continuación se describen los pasos para realizar el método

1. Sea (x_{ii}, y_{ii}) el punto al cual se le desea aproximar un valor z_{ii}

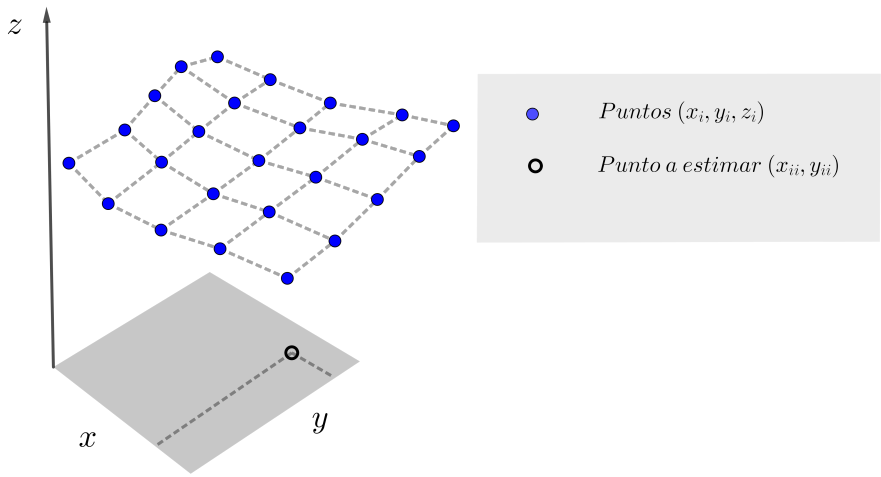


Figura 3.1

realizamos una serie de interpolaciones en el plano (x, z) para encontrar una serie de valores en z para el punto x_{ii}, llamaremos a esa serie de puntos z_{ip}

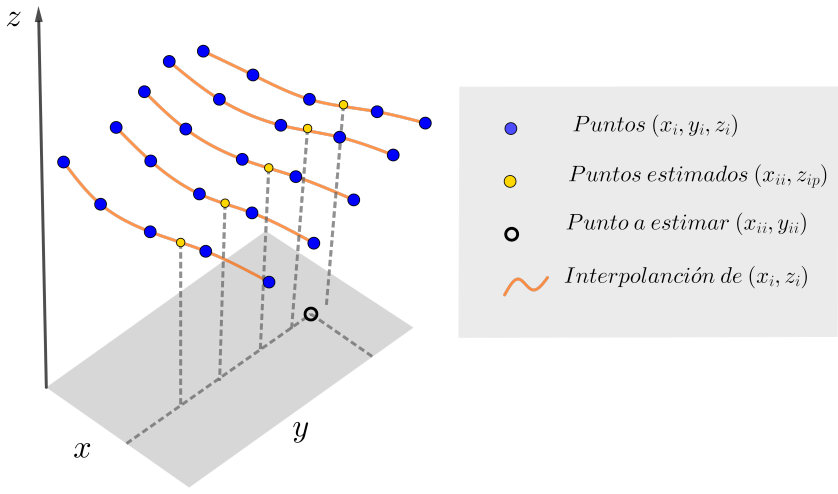


Figura 3.2

2. Una vez obtenida la serie de puntos z_{ip}, interpolamos en el plano (y, z) la serie de puntos (y_i, z_{ip}) para estimar el valor z_{ii} en y_{ii}

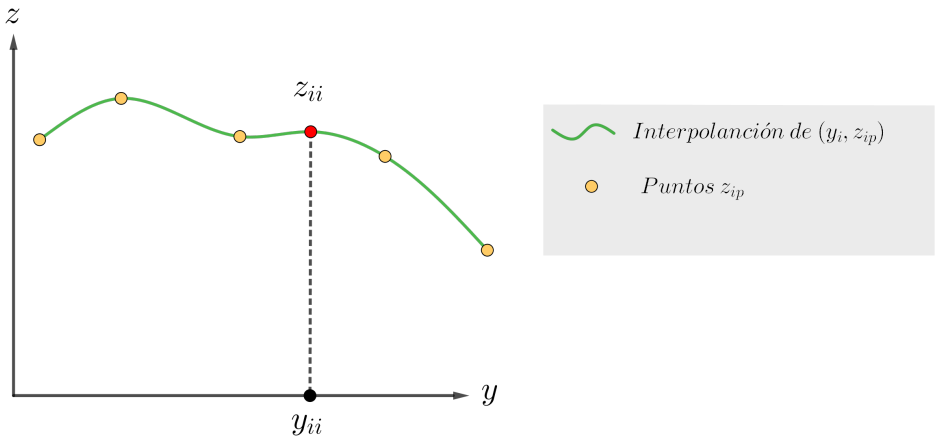


Figura 3.3

3. El punto z_{ii} estimado en el paso anterior es la aproximación bivariada del punto (x_{ii}, y_{ii}, z_{ii})

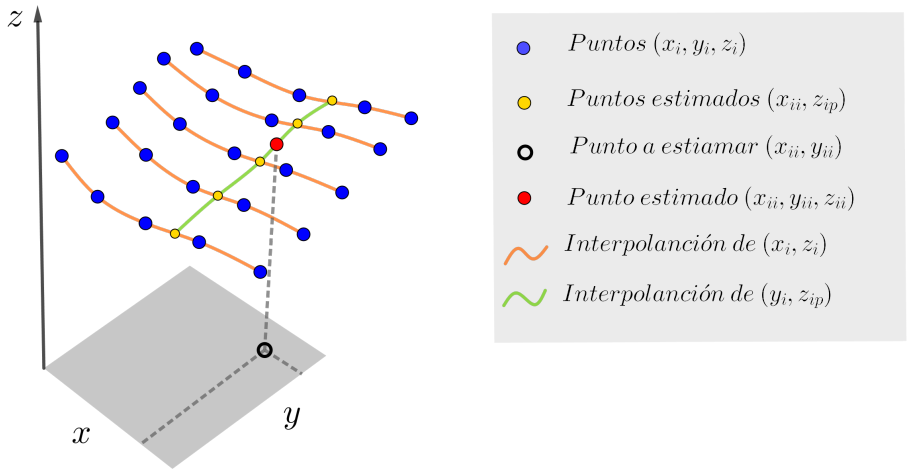


Figura 3.4

Es importante notar que todas las interpolaciones deben hacerse con el mismo método (Spline, Akima, etc.).

3.2. Interpolación Bivariada de Akima

En la sección anterior se usaron los interpolantes unidimensionales para realizar una interpolación bivarida en los puntos (x_i, y_i, z_i) y aproximar valores en la región formada, ahora se estudiará un método descrito en el siguiente artículo [4, pág. 15-25]., el cual consiste en realizar la interpolación de los puntos (x_i, y_i, z_i) creando una malla triangular con base a su posición y construyendo polinomios de grado cinco con la siguiente estructura

$$z(x, y) = \sum_{j=0}^{5} \sum_{k=0}^{5-j} q_{jk} x^j y^k$$

para cada triángulo formado

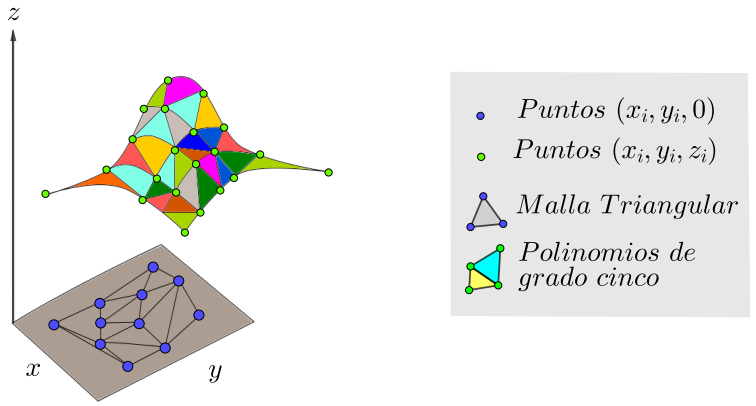


Figura 3.5

ahora veremos los pasos para encontrar los coeficientes adecuados para cada polinomio de la malla construida con base a nuestros puntos (x_i, y_i, z_i).

3.2.1. Triangulación

El primer paso para realizar la interpolación Bivariada de Akima, es construir una malla triangular uniendo solamente los puntos (x_i, y_i) en el dominio, mediante los siguientes pasos:

1. Determinamos el par de puntos más cercanos y dibujamos un segmento de linea entre ellos.

2. Buscamos en los puntos restantes, el par de puntos más cercanos y dibujamos un segmento de recta entre ellos siempre y cuando no cruce con algún segmento anterior.

3. Repetimos el paso 2 hasta agotar todos los posibles pares de puntos que puedan formarse.

A continuación se muestran dos gráficas que dan un ejemplo de la construcción de una malla triangular, dado un conjunto de puntos (x_i, y_i) distribuidos irregularmente.

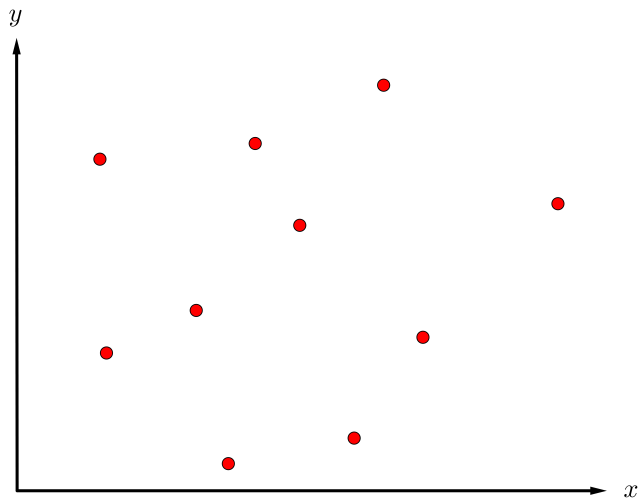


Figura 3.6: Conjunto de puntos (x_i, y_i) distribuidos irregularmente.

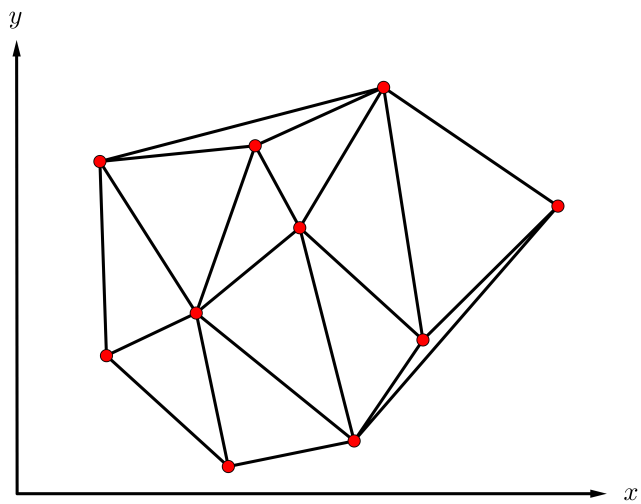


Figura 3.7: Triangulación de puntos (x_i, y_i).

3.2.2. Aproximación a las derivadas parciales

Para poder encontrar los coeficientes de los polinomios tal que formen en conjunto una superficie suave, es necesario tener una valor estimado de las derivadas parciales z_x,z_y,z_{xx},z_{xy} y z_{yy} para cada punto (x_i, y_i, z_i), como se muestra en las figuras

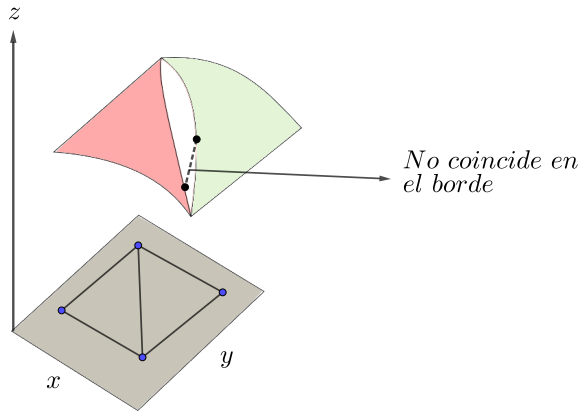


Figura 3.8: Superficie no suave (sin derivadas parciales).

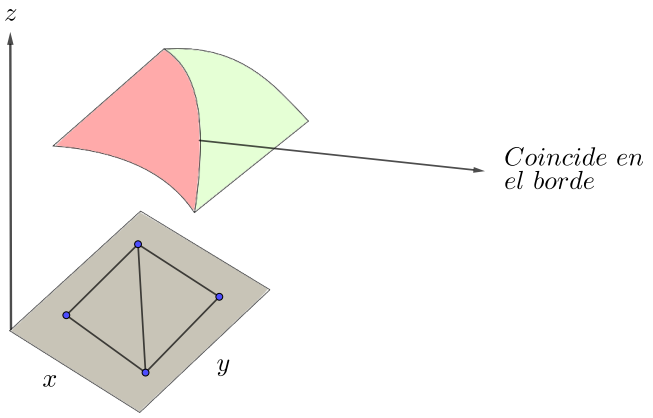


Figura 3.9: Superficie suave (con derivadas parciales).

a continuación se desarrollará un método para aproximar las derivadas parciales

1. Sea $P_i = (x_i, y_i, z_i)$ donde $i = 1, ..., n$, para P_1, buscamos las proyecciones más cercanas de cada P_i a la proyección de P_1, es decir, buscamos los puntos $(x_i, y_i, 0)$ para $i = 2, 3, .., n$ más cercanos a $(x_1, y_1, 0)$, se recomienda tomar de tres a cinco puntos, en este caso contemplaremos tres puntos, una vez obtenidas las proyecciones más cercanas, observamos a que punto P_i corresponden y denotamos P_1c,P_2c y P_2c para cada punto.

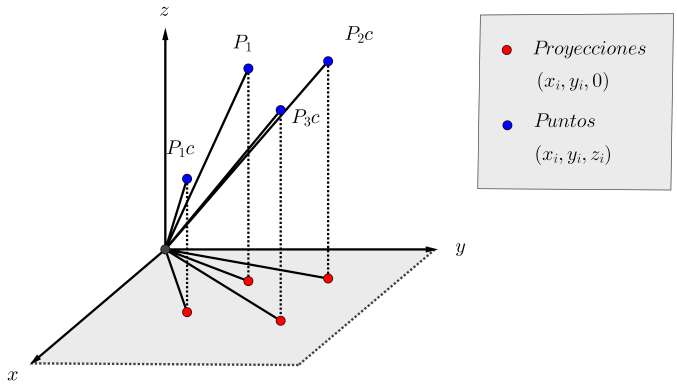


Figura 3.10: Punto P_1 y tres proyecciones más cercanas.

2. Obtenemos todos los vectores $u_j = P_{jc} - P_1$, para $j = 1, 2, 3$

$$u_1 = P_{1c} - P_1$$
$$u_2 = P_{2c} - P_1$$
$$u_3 = P_{3c} - P_1$$

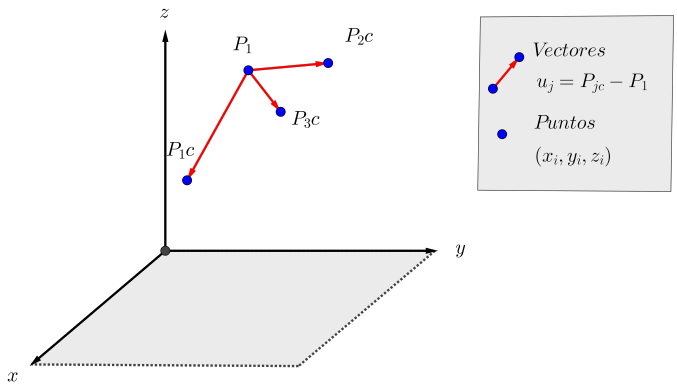


Figura 3.11: Vectores $u_j = P_{jc} - P1$ para $j = 1, 2, 3$.

3. Realizamos el producto vectorial de todas la combinaciones de u_1, u_2, u_3, siempre que la componente z sea positiva, si no es así, se intercambia el orden de los factores del producto, obteniendo:

$$v_1 = u_1 \times u_2 \text{ ó } v_1 = u_2 \times u_1$$
$$v_2 = u_2 \times u_3 \text{ ó } v_2 = u_3 \times u_2$$
$$v_3 = u_1 \times u_3 \text{ ó } v_3 = u_3 \times u_1$$

recordando que el producto vectorial de dos vectores a, b se define como

$$a \times b = \begin{vmatrix} I & J & K \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix}$$

donde

$$I = (1, 0, 0)$$
$$J = (0, 1, 0)$$
$$K = (0, 0, 1)$$

vectores canónicos unitarios.

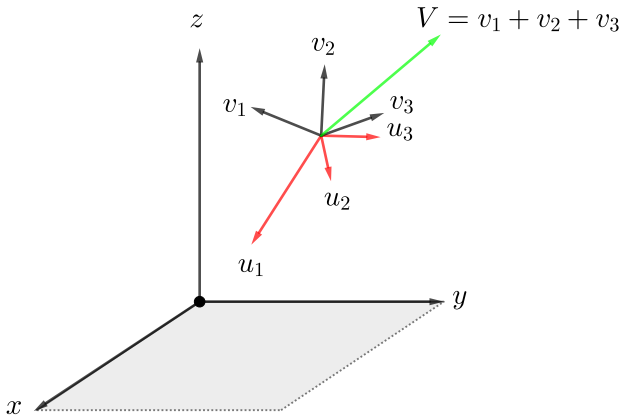


Figura 3.12: Vectores Producto de todas la combinaciones $u_i \times u_j$ con $i, j = 1, 2, 3$ e $i \neq j$ de componente z positiva.

4. Sumamos los vectores v_j para $j = 1, 2, 3$, sea $V = v_1 + v_2 + v_3 = (V_1, V_2, V_3)$ y determinamos z_x y z_y

$$z_{x1} = -\frac{V_1}{V_3}$$
$$z_{y1} = -\frac{V_2}{V_3}$$

y repetimos el proceso, a partir del Paso 1 hasta el Paso 4 en cada punto P_i donde $i = 1, 2, ..., n$ para determinar cada z_{xi} y z_{yi}.

5. Realizamos del Paso 1 hasta el Paso 4, pero ahora, en el punto $P_1 = (x_1, y_1, z_{x1})$ obteniendo

$$z_{xx1} = -\frac{V_1}{V_3}$$
$$z_{xyp1} = -\frac{V_2}{V_3}$$

después para cada punto $P_i = (x_i, y_i, z_{xi})$

6. Repetimos los pasos del 1-4 para $P_1 = (x_1, y_1, z_{y1})$ dando como resultado

$$z_{yxp1} = -\frac{V_1}{V_3}$$
$$z_{yy1} = -\frac{V_2}{V_3}$$

realizar ahora para $P_i = (x_i, y_i, z_{yi})$

7. Finalmente z_{xyi}

$$z_{xyi} = \frac{z_{xypi} + z_{yxpi}}{2}$$

Obteniendo así, una aproximación a las derivadas parciales $z_x, z_y, z_{xx}, z_{xy}, z_{yy}$. El mismo proceso se realiza para cuatro y cinco proyecciones cercanas, con lo cual aumenta en número de vectores u_j y v_j.

A continuación, se muestra una tabla con treinta valores y el cálculo de sus derivadas parciales z_x,z_y,z_{xx},z_{xy},z_{yy}.

Cuadro 3.1: Cálculo de z_x,z_y,z_{xx},z_{xy},z_{yy}.

x	y	z	z_x	z_y	z_{xx}	z_{xy}	z_{yy}
3.7927	5.3326	0.2950	-0.0520	0.0534	-0.0051	-0.0003	-0.0005
2.6975	48.0949	0.5029	0.0584	0.1630	-0.0054	-0.0215	-0.0351
26.5399	0.2317	0.9977	-0.4293	0.0551	0.0139	-0.0177	0.0203
38.9584	38.7455	0.7419	0.0182	0.0224	-0.0006	0.0006	-0.0177
46.7005	40.8652	-0.3884	-0.0904	-0.0880	-0.0122	-0.0103	-0.0124
6.4953	43.4347	-0.3292	0.1005	0.2002	-0.0067	-0.0216	-0.0343
28.4412	4.2218	0.9480	-0.5013	0.0109	0.0393	-0.0091	0.0213
23.4695	19.9891	-0.5000	0.1579	-0.1262	-0.0268	0.0269	-0.0307
0.5951	12.9935	0.8533	-0.0535	0.0481	-0.0055	-0.0010	0.0000
16.8561	40.0034	0.3059	0.0654	0.0062	-0.0021	-0.0058	0.0004
8.1091	21.5707	-0.9864	-0.4855	-0.2607	-0.0829	-0.0219	0.0016
39.7142	45.5324	-0.4110	-0.0554	-0.1328	-0.0070	-0.0062	-0.0203
15.5608	9.0924	-0.4615	-0.0012	0.0597	0.0228	-0.0082	-0.0302
26.4267	13.1901	0.9404	0.0016	0.2197	-0.0288	-0.0035	0.0269
8.2824	7.2769	0.1480	-0.0830	0.0587	0.0046	0.0000	-0.0009
30.0991	6.8034	-0.7150	-0.0335	-0.0080	0.0405	0.0240	0.0293
13.1486	43.4646	0.0645	0.0645	-0.0108	-0.0025	-0.0117	-0.0527
32.7040	28.9852	-0.9097	-0.1598	0.1982	0.0103	0.0068	-0.0165
34.4607	27.4930	-0.7695	0.1511	0.0751	0.0143	-0.0228	0.0072
37.4076	7.2477	0.6234	0.0984	-0.1692	0.0263	0.0321	0.0094
22.5271	42.6516	0.7137	0.0667	0.0017	-0.0013	-0.0049	-0.0137
4.1911	31.1028	-0.6716	0.0206	-0.0360	-0.0704	-0.0002	0.0146
11.4488	17.5476	-0.6610	-0.3531	-0.2529	-0.1223	-0.0971	-0.0749
45.6669	25.6625	0.8000	0.1514	0.0738	0.0096	-0.0107	-0.0024
7.6189	20.0904	0.5354	-0.0783	-0.1459	-0.0102	-0.0307	-0.0185
41.2908	3.7983	0.8943	-0.0008	-0.2933	0.0001	0.0033	0.0541
26.9171	11.9958	0.9370	0.0438	0.2844	-0.0228	0.0086	0.0499
49.8067	6.1659	-0.5447	-0.0375	-0.1208	-0.0030	0.0126	0.0359
3.9088	9.1954	0.5122	-0.0544	0.0502	-0.0058	-0.0007	-0.0002
22.1339	11.9976	0.4132	0.1087	0.0538	-0.0292	0.0686	-0.1178

3.2.3. Caracterización del polinomio

Una vez obtenida la malla triangular de los puntos y las derivadas parciales, procedemos a encontrar el valor de los coeficientes de cada polinomio, recordando la estructura

$$z(x,y) = \sum_{j=0}^{5} \sum_{k=0}^{5-j} q_{jk} x^j y^k \tag{3.1}$$

notemos que es necesario calcular 21 coeficientes para el polinomio de cada triángulo en la malla, los pasos a seguir son los siguientes

1. Para cada vértice del triángulo denotamos P_1,P_2 y P_3 respectivamente en sentido "contrario a las manecillas del reloj"

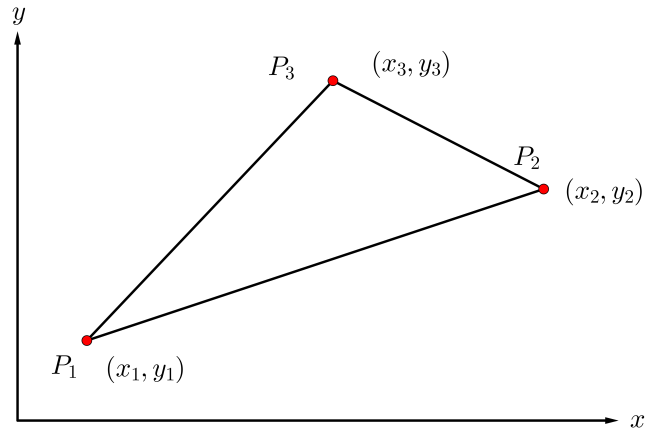


Figura 3.13: P_1,P_2 y P_3 en sentido "contrario a las manecillas del reloj".

2. Realizamos una transformación "u, v" en los puntos P_1,P_2 y P_3 como se muestra en la figura:

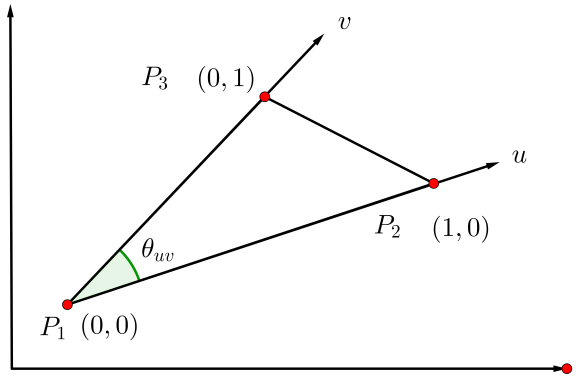
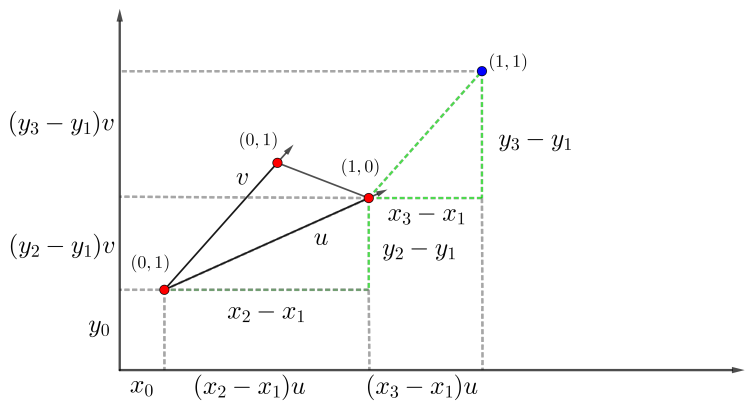


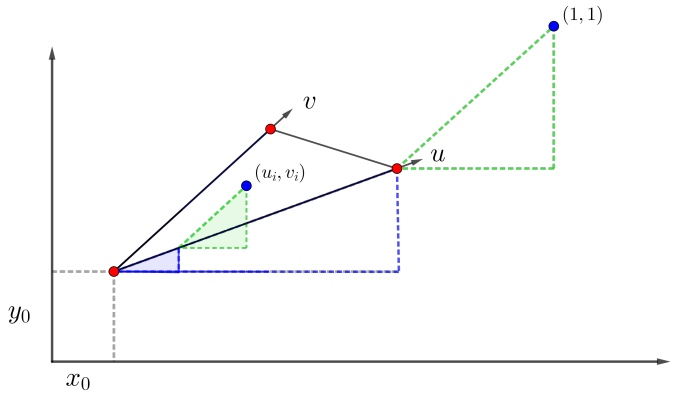
Figura 3.14: Transformación al sistema "u, v".

ahora para encontrar la expresión que corresponde para la nueva transformación, llevaremos un punto del sistema "u, v" al original,



podemos observar en la imagen anterior, los triángulos rectángulos cuya dimensión de hipotenusa corresponde a la longitud de las coordenadas (u, v) en ese sistema, en este caso $(1, 1)$, para obtener la coordenada "x" en el sistema, notemos que basta con sumar cada uno de los catetos de los triángulos rectángulos paralelos al eje x y al final sumar la distancia que hay en ese mismo eje entre orígenes de ambos sistemas. Conocemos la distancia de los catetos que son $x_2 - x_1$ y $x_3 - x_1$, ahora puesto que la distancia de u, v con su origen es 1, las distancias de los catetos se pueden reescribir $(x_2 - x_1)u$ y $(x_3 - x_1)v$ respectivamente, este mismo razonamiento nos lleva a encontrar el valor de las distancias en los catetos paralelos al eje y, cuyas expresiones son $(y_2 - y_1)u$ y $(y_3 - y_1)v$.

El procedimiento anterior y la imagen permiten notar la relación existente entre ambos sistemas, el mismo criterio es válido para cualquier puntos (u, v) ya que los triángulos formados serían semejantes a los del puntos $(1, 1)$ ya que ángulo entre la hipotenusa y el cateto sería siempre el de u y v respectivamente conservando la relación proporcional de ambos sistemas para cada cateto como se muestra en la imagen siguiente



retomando lo anterior podemos obtener los siguientes resultados

$$\begin{aligned} x &= au + bv + x_0 \\ y &= cu + dv + y_0 \end{aligned} \tag{3.2}$$

$$\begin{aligned} a &= x_2 - x_1 \\ b &= x_3 - x_1 \\ c &= y_2 - y_1 \\ d &= y_3 - y_1 \\ x_0 &= x_1 \\ y_0 &= y_1 \end{aligned} \tag{3.3}$$

y despejando u y v de 3.2

$$\begin{aligned} u &= \frac{d(x - x_0) - b(y - y_0)}{ad - bc} \\ v &= \frac{-c(x - x_0) + a(y - y_0)}{ad - bc} \end{aligned} \tag{3.4}$$

Recordando la regla de la cadena generalizada para una transformación

$$\frac{\partial z}{\partial u} = \frac{\partial z}{\partial x}\frac{\partial x}{\partial u} + \frac{\partial z}{\partial y}\frac{\partial y}{\partial u}$$

$$\frac{\partial z}{\partial v} = \frac{\partial z}{\partial x}\frac{\partial x}{\partial v} + \frac{\partial z}{\partial y}\frac{\partial y}{\partial v}$$

$$\frac{\partial^2 z}{\partial u^2} = \frac{\partial z}{\partial x}\frac{\partial^2 x}{\partial u^2} + \frac{\partial^2 z}{\partial x^2}\left(\frac{\partial x}{\partial u}\right)^2 + \cdots$$

$$\cdots + \frac{\partial z}{\partial y}\frac{\partial^2 y}{\partial u^2} + \frac{\partial^2 z}{\partial y^2}\left(\frac{\partial y}{\partial u}\right)^2 + 2\frac{\partial^2 z}{\partial x \partial y}\frac{\partial x}{\partial u}\frac{\partial y}{\partial u}$$

$$\frac{\partial^2 z}{\partial u \partial v} = \frac{\partial z}{\partial x}\frac{\partial^2 x}{\partial u \partial v} + \frac{\partial^2 z}{\partial x^2}\frac{\partial x}{\partial u}\frac{\partial x}{\partial v} + \cdots$$

$$\cdots + \frac{\partial z}{\partial y}\frac{\partial^2 y}{\partial u \partial v} + \frac{\partial^2 z}{\partial y^2}\frac{\partial y}{\partial u}\frac{\partial y}{\partial v} + \frac{\partial^2 z}{\partial x \partial y}\frac{\partial x}{\partial v}\frac{\partial y}{\partial u} + \frac{\partial^2 z}{\partial y \partial x}\frac{\partial x}{\partial u}\frac{\partial y}{\partial v}$$

$$\frac{\partial^2 z}{\partial v^2} = \frac{\partial z}{\partial x}\frac{\partial^2 x}{\partial v^2} + \frac{\partial^2 z}{\partial x^2}\left(\frac{\partial x}{\partial v}\right)^2 + \cdots$$

$$\cdots + \frac{\partial z}{\partial y}\frac{\partial^2 y}{\partial v^2} + \frac{\partial^2 z}{\partial y^2}\left(\frac{\partial y}{\partial v}\right)^2 + 2\frac{\partial^2 z}{\partial x \partial y}\frac{\partial x}{\partial v}\frac{\partial y}{\partial v}$$

[7, pág. 242-269], sustituyendo con 3.2

$$z_u = z_x\frac{\partial au + bv + x_0}{\partial u} + z_y\frac{\partial cu + dv + y_0}{\partial u} = az_x + cz_y$$

$$z_v = z_x\frac{\partial av + bv + x_0}{\partial v} + z_y\frac{\partial cv + dv + y_0}{\partial v} = bz_x + dz_y$$

procediendo de la misma forma en las últimas tres ecuaciones en la regla

de la cadena, se obtiene

$$z_u = az_x + cz_y$$

$$z_v = bz_x + dz_y$$

$$z_{uu} = a^2 z_{xx} + 2acz_{xy} + c^2 z_{yy} \tag{3.5}$$

$$z_{uv} = abz_{xx} + (ad + bc)z_{xy} + cdz_{yy}$$

$$z_{vv} = b^2 z_{xx} + 2bdz_{xy} + d^2 z_{yy}$$

Realizando la transformación, el polinomio tiene la misma estructura de 3.1

$$z(u, v) = \sum_{j=0}^{5} \sum_{k=0}^{5-j} P_{jk} u^j v^k \tag{3.6}$$

pero ahora se busca encontrar los valores de P_{jk} adecuados para obtener el interpolante, con base a lo anterior calculamos las nuevas derivadas parciales de 3.6

$$\begin{aligned}
z_u(u, v) &= \sum_{j=1}^{5} \sum_{k=0}^{5-j} jP_{jk} u^{j-1} v^k \\
z_v(u, v) &= \sum_{j=0}^{4} \sum_{k=1}^{5-j} kP_{jk} u^j v^{k-1} \\
z_{uu}(u, v) &= \sum_{j=2}^{5} \sum_{k=0}^{5-j} j(j-1)P_{jk} u^{j-2} v^k \\
z_{uv}(u, v) &= \sum_{j=1}^{4} \sum_{k=1}^{5-j} jkP_{jk} u^{j-1} v^{k-1} \\
z_{vv}(u, v) &= \sum_{j=0}^{3} \sum_{k=2}^{5-j} k(k-1)P_{jk} u^j v^{k-2}
\end{aligned} \tag{3.7}$$

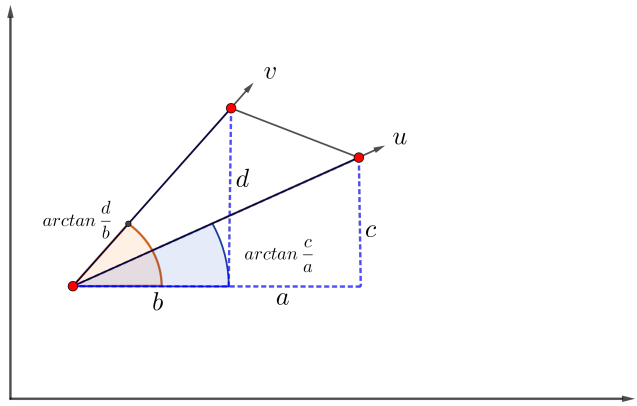
Denotamos la longitud del segmento $\overline{P_1P_2}$, $\overline{P_1P_3}$ y el ángulo θ_{uv} como sigue

$$L_u = a^2 + c^2$$

$$L_v = b^2 + d^2 \tag{3.8}$$

$$\theta_{uv} = \arctan \frac{d}{b} - \arctan \frac{c}{a}$$

para encontrar el valor anterior, se trazan los siguientes triángulos rectángulos

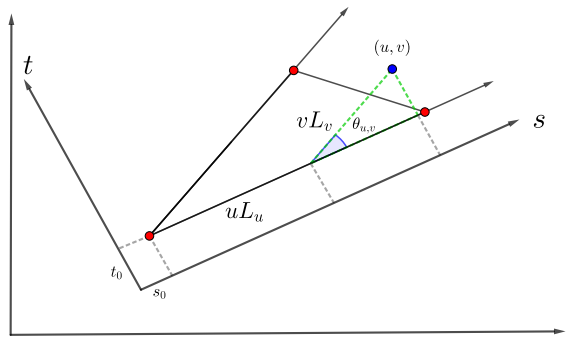


usando la notación de 3.3 y las razones trigonométricas, obtenemos que el valor del ángulo en cada triángulo es arctan $\frac{d}{b}$ y arctan $\frac{c}{a}$, en la imagen anterior notemos que es suficiente con la diferencia entre ambos ángulos para obtener el valor de $\theta_{u,v}$.

3. Realizamos la transformación "s,t"
Ahora debemos hacer una nueva transformación "s,t" con el propósito de crear sistemas de ecuaciones que permitan encontrar los coeficientes P_{jk} del polinomio de la transformación 3.6, para realizarlo se desarrollara el procedimiento en tres fases, rotando el eje "s" paralelamente a cada segmento que forma el triángulo, un nuevo supuesto para esta transformación es el siguiente.

$$z(u,v)_{tssss} = 0 \tag{3.9}$$

En la figura 3.15 debemos encontrar la relación entre ambos sistemas, para lograrlo trazamos un triángulo rectángulo cuyo ángulo entre el cateto paralelo al eje "s" y la hipotenusa es $\theta_{u,v}$ y además pase por el punto (u,v)



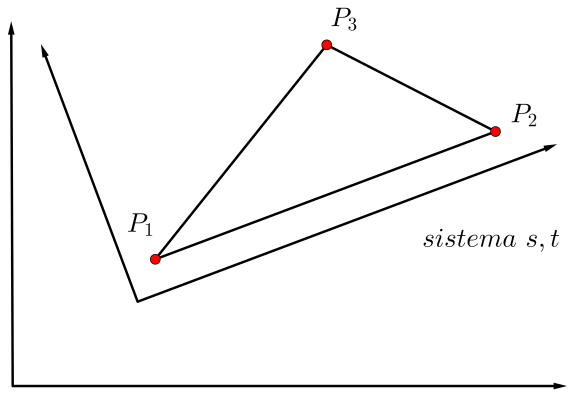
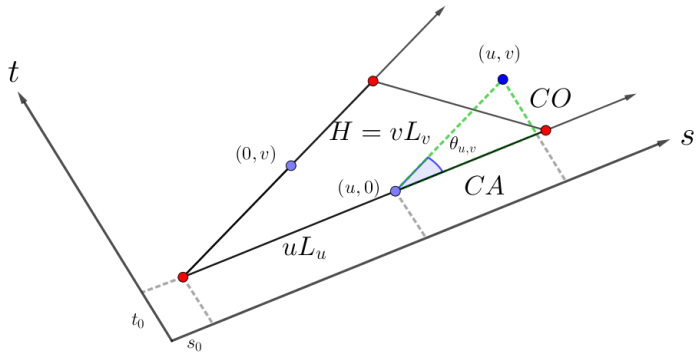


Figura 3.15: Sistema "s, t" paralelo a $\overline{P_1P_2}$.

nuevamente, para obtener el valor s en el nuevo sistema, tenemos que sumar las distancias de s_0 que es la distancia en el eje "s" de los orígenes en cada sistema, y las distancias de (u, v) llevado al sistema "t, s", dado que el eje "s" es paralelo al eje "u" entonces basta con el producto uL_u ya que representa la proporción adecuada para el eje "s",



el valor de la longitud en la hipotenusa del triángulo trazado es vLv, finalmente para encontrar el valor del cateto que nos falta podemos usar la relación

$$\cos\theta_{u,v} = \frac{Cateto\ Adyacente}{vL_v}$$

obteniendo así

$$Cateto\ Adyacente = \cos\theta_{u,v} v L_v$$

de manera similar obtenemos el valor del cateto paralelo al eje "t"

$$Cateto\ Opuesto = \sin\theta_{u,v} v L_v$$

con todo lo anterior podemos dar una expresión de la relación entre ambos sistemas

$$s = vL_v \cos\theta_{u,v} + uL_u + s_0$$

$$t = vL_v \sin\theta_{u,v} + t_0$$

en consecuencia obtenemos

$$\begin{aligned} u &= \frac{\sin(\theta_{uv})(s - s_0) - \cos(\theta_{uv})(t - t_0)}{L_u \sin(\theta_{uv})} \\ v &= \frac{t - t_0}{L_v \sin(\theta_{uv})} \end{aligned} \tag{3.10}$$

para facilitar los cálculos de la derivada en el polinomio $z(u, v)$ calculamos las derivadas parciales de s y t en términos de u y v, tomando en cuenta la ecuación 3.10 con la regla de la cadena antes vista

$$\begin{aligned} \frac{\partial}{\partial s} &= \frac{\partial u}{\partial s}\frac{\partial}{\partial u} + \frac{\partial v}{\partial s}\frac{\partial}{\partial v} = \frac{1}{L_u}\frac{\partial}{\partial u} \\ \frac{\partial}{\partial t} &= \frac{\partial u}{\partial t}\frac{\partial}{\partial u} + \frac{\partial v}{\partial t}\frac{\partial}{\partial v} = \frac{\cos(\theta_{uv})}{L_u \sin(\theta_{uv})}\frac{\partial}{\partial u} + \frac{1}{L_v \sin(\theta_{uv})}\frac{\partial}{\partial v} \end{aligned} \tag{3.11}$$

usando 3.6,3.9 y 3.11 el cálculo de $z(u, v)_{tssss}$ es de la siguiente forma

$$z(u, v)_{tssss} =$$

$$\frac{1}{L_u^4}\frac{\partial}{\partial u}\frac{\partial}{\partial u}\frac{\partial}{\partial u}\frac{\partial}{\partial u}\left(\frac{\cos(\theta_{uv})}{L_u \sin(\theta_{uv})}\frac{\partial}{\partial u}z(u, v) + \frac{1}{L_v \sin(\theta_{uv})}\frac{\partial}{\partial v}z(u, v)\right) = 0$$

finalmente, al realizar la operación se tiene

$$L_u P_{41} - 5L_v \cos(\theta_{uv}) P_{50} = 0 \tag{3.12}$$

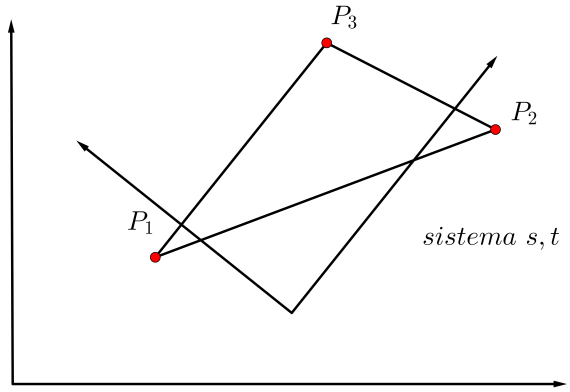


Figura 3.16: Sistema "s,t" paralelo a $\overline{P_1P_3}$.

siguiendo todo el razonamiento anterior para las dos transformaciones restantes, de la figura 3.16 resulta

$$\begin{aligned} u &= \frac{-(t-t_0)}{L_u \sin(\theta_{uv})} \\ v &= \frac{\sin(\theta_{uv})(s-s_0) + \cos(\theta_{uv})(t-t_0)}{L_v \sin(\theta_{uv})} \end{aligned} \tag{3.13}$$

donde las derivadas parciales son

$$\begin{aligned} \frac{\partial}{\partial s} &= \frac{\partial u}{\partial s}\frac{\partial}{\partial u} + \frac{\partial v}{\partial s}\frac{\partial}{\partial v} = \frac{1}{L_v}\frac{\partial}{\partial v} \\ \frac{\partial}{\partial t} &= \frac{\partial u}{\partial t}\frac{\partial}{\partial u} + \frac{\partial v}{\partial t}\frac{\partial}{\partial v} = \frac{1}{L_u \sin(\theta_{uv})}\frac{\partial}{\partial u} + \frac{\cos(\theta_{uv})}{L_v \sin(\theta_{uv})}\frac{\partial}{\partial v} \end{aligned} \tag{3.14}$$

con 3.6,3.9 y 3.14

$$L_v P_{14} - 5L_u \cos(\theta_{uv}) P_{05} = 0 \tag{3.15}$$

para la figura 3.17 se tiene

$$\begin{aligned} u &= A(s-s_0) + B(t-t_0) \\ v &= C(s-s_0) + D(t-t_0) \end{aligned} \tag{3.16}$$

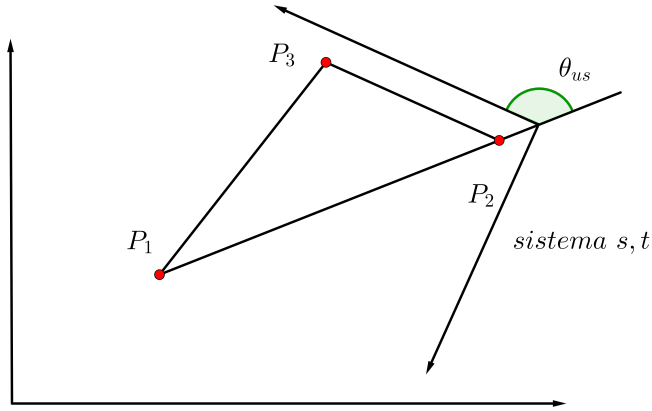


Figura 3.17: Sistema "s,t" paralelo a $\overline{P_2P_3}$.

donde

$$A = \frac{\sin(\theta_{uv} - \theta_{us})}{L_u \sin\theta_{uv}}$$

$$B = \frac{-\cos(\theta_{uv} - \theta_{us})}{L_u \sin\theta_{uv}}$$

$$C = \frac{\sin(\theta_{us})}{L_v \sin(\theta_{uv})} \tag{3.17}$$

$$D = \frac{\cos(\theta_{us})}{L_v \sin(\theta_{uv})}$$

$$\theta_{us} = \arctan\frac{d-c}{b-a} - \arctan\frac{c}{a}$$

por lo tanto las derivadas parciales son

$$\begin{aligned} \frac{\partial}{\partial s} &= \frac{\partial u}{\partial s}\frac{\partial}{\partial u} + \frac{\partial v}{\partial s}\frac{\partial}{\partial v} = A\frac{\partial}{\partial u} + C\frac{\partial}{\partial v} \\ \frac{\partial}{\partial t} &= \frac{\partial u}{\partial t}\frac{\partial}{\partial u} + \frac{\partial v}{\partial t}\frac{\partial}{\partial v} = B\frac{\partial}{\partial u} + D\frac{\partial}{\partial v} \end{aligned} \tag{3.18}$$

retomando 3.6,3.9 y 3.18

$$5A^4BP_{50} + A^3(4BC + AD)P_{41} +$$

$$A^2C(3BC + 2AD)P_{32} + AC^2(2BC + 3AD)P_{23} + \tag{3.19}$$

$$C^3(BC + 4AD)P_{14} + 5C^4DP_{05} = 0$$

4. Calculamos los coeficientes del polinomio de acuerdo a 3.6 y 3.7 con lo cual obtenemos los siguientes resultados

$$\begin{gathered} z(0,0) = P_{00} \\ z_u(0,0) = P_{10} \\ z_v(0,0) = P_{01} \\ \frac{z_{uu}(0,0)}{2} = P_{20} \\ z_{uv}(0,0) = P_{11} \\ \frac{z_{vv}(0,0)}{2} = P_{02} \end{gathered} \tag{3.20}$$

hacemos $u = 1$ y $v = 0$ en las primeras tres ecuaciones 3.7

$$P_{30} + P_{40} + P_{50} = z(1,0) - P_{00} - P_{10} - P_{20}$$

$$3P_{30} + 4P_{40} + 5P_{50} = z_u(1,0) - P_{10} - 2P_{20}$$

$$6P_{30} + 12P_{40} + 20P_{50} = z_{uu}(1,0) - 2P_{20}$$

resolvemos para P_{00},P_{40},P_{50}

$$\begin{gathered} P_{30} = \frac{20z(1,0) - 8z_u(1,0) + z_{uu}(1,0) - 20P_{00} - 12P_{10} - 6P_{20}}{2} \\ P_{40} = -15z(1,0) + 7z_u(1,0) - z_{uu}(1,0) + 15P_{00} + 8P_{10} + 3P_{20} \\ P_{50} = \frac{12z(1,0) - 6z_u(1,0) + z_{uu}(1,0) - 12P_{00} - 6P_{10} - 2P_{20}}{2} \end{gathered} \tag{3.21}$$

de la misma forma usando z, z_u y z_{vv} haciendo $u = 0$ y $v = 1$ en las ecuaciones 3.6 y 3.7 podemos obtener los siguientes resultados

$$P_{03} = \frac{20z(0,1) - 8z_v(0,1) + z_{vv}(0,1) - 20P_{00} - 12P_{01} - 6P_{02}}{2}$$

$$P_{04} = -15z(0,1) + 7z_v(0,1) - z_{vv}(0,1) + 15P_{00} + 8P_{01} + 3P_{02}$$

$$P_{05} = \frac{12z(0,1) - 6z_v(0,1) + z_{vv}(0,1) - 12P_{00} - 6P_{01} - 2P_{02}}{2} \tag{3.22}$$

hemos obtenido P_{50} y P_{05} ahora podemos encontrar P_{41} y P_{14} de 3.12 y 3.15

$$P_{41} = \frac{5L_v \cos(\theta_{uv})}{L_u} P_{50}$$

$$P_{14} = \frac{5L_u \cos(\theta_{uv})}{L_v} P_{05} \tag{3.23}$$

tomamos z_u y z_{uv} de 3.7 con $u = 1$ y $v = 0$ llegando a

$$P_{21} + P_{31} = z_v(1,0) - P_{01} - P_{11} - P_{41}$$

$$2P_{21} + 3P_{31} = z_{uv}(1,0) - P_{11} - 4P_{41}$$

y resolviendo para P_{21} y P_{31}

$$P_{21} = 3z_v(1,0) - z_{uv}(1,0) - 3P_{01} - 2P_{11} + P_{41}$$

$$P_{31} = -2z_v(1,0) + z_{uv}(1,0) + 2P_{01} + P_{11} - 2P_{41} \tag{3.24}$$

de manera similar usamos z_u y z_{uv} de 3.7 con $u = 0$ y $v = 1$ dando como resultado

$$P_{12} = 3z_u(0,1) - z_{uv}(0,1) - 3P_{10} - 2P_{11} + P_{14}$$

$$P_{13} = -2z_u(0,1) + z_{uv}(0,1) + 2P_{10} + P_{11} - 2P_{14} \tag{3.25}$$

podemos reescribir 3.19 como

$$g_1 P_{32} + g_2 P_{23} = h_1 \tag{3.26}$$

donde

$$
\begin{aligned}
g_1 &= A^2C(3BC + 2AD) \\
g_2 &= AC^2(2BC + 3AD) \\
h_1 &= -5A^4BP_{50} - A^3(4BC + AD)P_{41} - \\
&\quad C^3(BC + 4AC)P_{41} - 5C^4DP_{05}
\end{aligned}
\tag{3.27}
$$

para z_{vv} de 3.7 y $u = 1$ y $v = 0$ obtenemos

$$P_{22} + P_{32} = h_2 \tag{3.28}$$

siendo

$$h_2 = \frac{1}{2}z_{vv}(1, 0) - P_{02} - P_{12} \tag{3.29}$$

de igual forma con z_{uu} de 3.7 y $u = 0$ y $v = 1$

$$P_{22} + P_{23} = h_3 \tag{3.30}$$

$$h_3 = \frac{1}{2}z_{uu}(0, 1) - P_{20} - P_{21} \tag{3.31}$$

formando el sistema de ecuaciones con 3.26,3.28 y 3.30

$$
\begin{aligned}
g_1P_{32} + \quad g2P_{23} &= h_1 \\
P_{22} + \quad P_{32} &= h_2 \\
P_{22} + \quad P_{23} &= h_3
\end{aligned}
$$

resolviendo para P_{22}, P_{23} y P_{32}

$$
\begin{aligned}
P_{22} &= \frac{g_1h_2 + g_2h_3 - h_1}{g_1 + g_2} \\
P_{32} &= h_2 - P_{22} \\
P_{23} &= h_3 - P_{22}
\end{aligned}
\tag{3.32}
$$

obteniendo así los veintiún coeficientes del polinomio 3.6, para llevar los valores al plano "x, y, z" basta con regresar a la transformación de 3.2.

En este capítulo hemos visto dos métodos de interpolación bivariada, por un lado los IBMIU teniendo como base los interpolantes univariados y la Interpolación Bivariada de Akima, ambos métodos ofrecen aproximaciones sobre superficies, ahora es necesario analizar cual de los métodos se ajusta mejor a los datos y el costo computacional de los interpolantes.

Capítulo 4

Descripción de Rutinas

A continuación se da una descripción de los códigos realizados en el software numérico Matlab en su versión 2016a utilizados para el cálculo de los interpolantes, los programas forman parte de dos interfaces gráficas que realizan la interpolación bivariada mediante los métodos descritos en el capítulo 3.

4.1. Reseña de Códigos

4.1.1. Interpolación Univariada

ILagrange

Calcula la interpolación de Lagrange mediante los siguientes parámetros

[newy]=ILagrange(x,y,newx)

1. Parámetros de Entrada:

   a) x,y:Vectores que contienen los puntos (x_i, y_i) para el cálculo del interpolante.

   b) newx: Vector que contiene los puntos que evaluará el método.

2. Parámetros de Salida:

   a) newy: Vector con los resultados al evaluar "newx" con el programa.

INewton

Calcula la interpolación de Newton mediante los siguientes parámetros

[newy]=INewton(x,y,newx)

1. Parámetros de Entrada:
   a) x,y:Vectores que contienen los puntos (x_i, y_i) para el cálculo del interpolante.
   b) newx: Vector que contiene los puntos que evaluará el método.
2. Parámetros de Salida:
   a) newy: Vector con los resultados al evaluar "newx" con el programa.

IHermite

Calcula la interpolación de Hermite mediante los siguientes parámetros

[newy]=IHermite(x,y,dy,newx)

1. Parámetros de Entrada:
   a) x,y:Vectores que contienen los puntos (x_i, y_i) para el cálculo del interpolante.
   b) dy: Vector que contiene el valor de la derivada en cada punto (x_i, y_i).
   c) newx: Vector que contiene los puntos que evaluará el método.
2. Parámetros de Salida:
   a) newy: Vector con los resultados al evaluar "newx" con el programa.

ILineal

Calcula la interpolación Lineal mediante los siguientes parámetros

[newy]=ILineal(x,y,newx)

1. Parámetros de Entrada:
   a) x,y:Vectores que contienen los puntos (x_i, y_i) para el cálculo del interpolante.
   b) newx: Vector que contiene los puntos que evaluará el método.
2. Parámetros de Salida:
   a) newy: Vector con los resultados al evaluar "newx" con el programa.

SCNatural

Calcula el interpolante Spline Cúbico Natural mediante los siguientes parámetros

[newy]=SCNatural(x,y,newx)

1. Parámetros de Entrada:

   a) x,y:Vectores que contienen los puntos (x_i, y_i) para el cálculo del interpolante.

   b) newx: Vector que contiene los puntos que evaluará el método.

2. Parámetros de Salida:

   a) newy: Vector con los resultados al evaluar "newx" con el programa.

SCCompleto

Calcula el interpolante Spline Cúbico Completo mediante los siguientes parámetros

[newy]=SCCompleto(x,y,dfa,dfb,newx)

1. Parámetros de Entrada:

   a) x,y:Vectores que contienen los puntos (x_i, y_i) para el cálculo del interpolante.

   b) dfa,dfb: Valor de la derivada en el punto (x_0, y_0) y (x_n, y_n) respectivamente.

   c) newx: Vector que contiene los puntos que evaluará el método.

2. Parámetros de Salida:

   a) newy: Vector con los resultados al evaluar "newx" con el programa.

SCNKnot

Calcula el interpolante Spline Cúbico Not-A-Knot mediante los siguientes parámetros

[newy]=SCNKnot(x,y,newx)

1. Parámetros de Entrada:

   a) x,y:Vectores que contienen los puntos (x_i, y_i) para el cálculo del interpolante.

   b) newx: Vector que contiene los puntos que evaluará el método.

2. Parámetros de Salida:

   a) newy: Vector con los resultados al evaluar "newx" con el programa.

ICAkima

Calcula el interpolante cúbico de Akima mediante los siguientes parámetros

[newy]=ICAkima(x,y,newx)

1. Parámetros de Entrada:

   a) x,y:Vectores que contienen los puntos (x_i, y_i) para el cálculo del interpolante.

   b) newx: Vector que contiene los puntos que evaluará el método.

2. Parámetros de Salida:

   a) newy: Vector con los resultados al evaluar "newx" con el programa.

ICCons

Calcula el interpolante cúbico Restringido mediante los siguientes parámetros

[newy]=ICCons(x,y,newx)

1. Parámetros de Entrada:

   a) x,y:Vectores que contienen los puntos (x_i, y_i) para el cálculo del interpolante.

b) newx: Vector que contiene los puntos que evaluará el método.

2. Parámetros de Salida:

a) newy: Vector con los resultados al evaluar "newx" con el programa.

MPIH

Calcula el interpolante cúbico Monótono mediante los siguientes parámetros

[newy]=MPIH(x,y,newx)

1. Parámetros de Entrada:

a) x,y:Vectores que contienen los puntos (x_i, y_i) para el cálculo del interpolante.

b) newx: Vector que contiene los puntos que evaluará el método.

2. Parámetros de Salida:

a) newy: Vector con los resultados al evaluar "newx" con el programa.

4.1.2. Interpolación Bivariada de Akima

Las siguientes rutinas están basadas en los códigos en Fortran 90 e implementadas en Matlab 2016b.[4, pág. 33-50]

idcldp

Determina los puntos cercanos de un conjunto irregularmente distribuido

[ipc]=idcldp(xd,yd,ncp)

1. Parámetros de Entrada:

a) xd,yd:Vectores que contienen los puntos (x_i, y_i) irregularmente distribuidos.

b) ncp: Número de puntos cercanos que se determinarán,es recomendado de tres a cinco puntos.

2. Parámetros de Salida:

a) ipc: Vector que contiene el índice de los puntos cercanos.

idpdrv

Aproxima las derivadas parciales z_x, z_y, z_{xx}, z_{xy} y z_{yy}

[pd]=idpdrv(xd,yd,zd,ncp,ipc)

1. Parámetros de Entrada:

   a) xd,yd,zd:Vectores que contienen los puntos (x_i, y_i, z_i) irregularmente distribuidos.

   b) ncp: Número de puntos cercanos que se determinarán,es recomendado de tres a cinco puntos.

   c) ipc: Índice de puntos cercanos

2. Parámetros de Salida:

   a) pd: Vector que contiene la aproximación a las derivadas parciales z_x, z_y, z_{xx}, z_{xy} y z_{yy}.

idtang

Programa que crea una malla triangular con puntos distribuidos irregularmente.

[nt,ipt,nl,ipl,iwl,iwp,wk]=idtang(xd,yd)

1. Parámetros de Entrada:

   a) xd,yd:Vectores que contienen los puntos (x_i, y_i) irregularmente distribuidos.

2. Parámetros de Salida:

   a) nt: Número de triángulos de la malla.

   b) ipt: Índice de los puntos que conforman los vértices del triángulo.

   c) nl: Número de lineas que conforman la malla triángular.

   d) ipl:Índice de los puntos que forman las lineas de la malla.

   e) iwl,iwp,wk: Vectores que contienen información de la malla.

idlctn

Función para encontrar un punto dentro de la malla triangular.

[iti,iwk, wk] =idlctn(xd, yd, nt, ipt, nl, ipl, xii, yii,iwk,wk)

1. Parámetros de Entrada:

   a) xd,yd:Vectores que contienen los puntos (x_i, y_i) irregularmente distribuidos.

   b) nt: Número de triángulos de la malla.

   c) ipt: Índice de los puntos que conforman los vértices del triángulo.

   d) nl: Número de lineas que conforman la malla triangular.

   e) ipl:Índice de los puntos que forman las lineas de la malla.

   f) xii,yii:Punto (x_k, y_k) que se buscará dentro de la malla triangular.

   g) iwk,wk: Vectores que contienen información de la malla.

2. Parámetros de Salida:

   a) iti: Índice del triángulo que contiene al punto.

   b) iwk,wk: Vectores que guardan la información de la búsqueda del punto en la malla.

idptip

Calcula el valor del interpolante en un punto.

[zii] = idptip(xd,yd,zd,nt,ipt,nl,ipl,pdd,iti,xii,yii)

1. Parámetros de Entrada:

   a) xd,yd,zd:Vectores que contienen los puntos (x_i, y_i, z_i) irregularmente distribuidos.

   b) nt: Número de triángulos de la malla.

   c) ipt: Índice de los puntos que conforman los vértices del triángulo.

   d) nl: Número de lineas que conforman la malla triangular.

   e) ipl:Índice de los puntos que forman las lineas de la malla.

   f) pdd:Vector que contiene la aproximación a las derivadas parciales z_x, z_y, z_{xx}, z_{xy} y z_{yy}.

g) iti: Índice del triángulo que contiene al punto.

h) xii,yii:Punto (x_k, y_k) que se evaluará en el interpolante.

2. Parámetros de Salida:

a) zii:Valor del interpolante en el punto (x_{ii}, y_{ii}).

idbvip

Calcula el valor de interpolante en una serie de puntos .

[zi,iwk,wk] = idbvip(ncp,xd,yd,zd,xi,yi)

1. Parámetros de Entrada:

a) xd,yd,zd:Vectores que contienen los puntos (x_i, y_i, z_i) irregularmente distribuidos.

b) xii,yii:Vector que contiene los puntos (x_k, y_k) que evaluará en el interpolante.

2. Parámetros de Salida:

a) zi:Valor del interpolante en los puntos (x_i, y_i).

b) iwk,wk: Información de la interpolacion en la serie de puntos.

3. Programas utilizados internamente

a) idcldp

b) idpdrv

c) idtang

d) idlctn

e) idptip

4.2. Interfaz Gráfica

Con el propósito de facilitar el uso de los programas previamente descritos se crean dos interfaces gráficas, en esta sección aprenderemos su funcionamiento, un interfaz interpola mediante el método de IBMIU, mientras que la segunda, interpola mediante la Interpolación Bivariada de Akima.
Es importante especificar que la interfaz trabaja con archivos de texto con

extensión .txt, delimitado por tabuladores, donde la primera fila del archivo contiene los caracteres del nombre de cada columna, ambas interfaces reciben dos archivos de texto, uno contiene los valores numéricos de (x, y, z) (tres columnas) que serán los puntos sobre con los cuales se realizará la interpolación, el segundo, contienen los puntos (x, y) que se aproximarán mediante la interpolación.

4.2.1. Interfaz de IBMIU

A continuación se muestra la imagen de la interfaz, que servirá para la descripción de su funcionamiento.

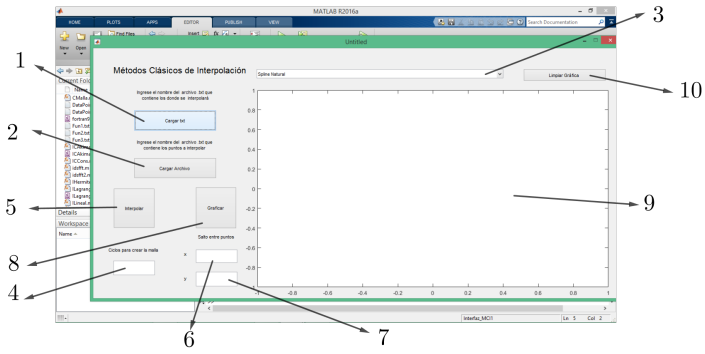


Figura 4.1: Interfaz para IBMIU

1. El botón permite cargar un archivo con extensión .txt con las tres columnas, que representan las coordenadas (x, y, z) que se utilizarán para la interpolación, la primera fila de estos datos deben contener el nombre de cada columna.

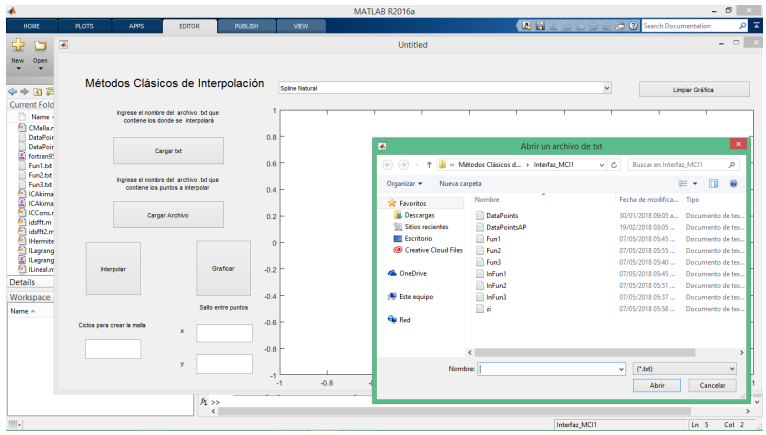


Figura 4.2: Interfaz para IBMIU

2. De manera similar que 1 permite cargar una archivo .txt, el cual debe contener dos columnas que son los puntos (x, y) que se aproximarán mediante el método elegido.

3. La pestaña permite escoger cual método univariado se utilizará para la interpolación bivariada.

4. Matlab maneja los datos de una malla mediante matrices, puesto que la interfaz acepta únicamente datos en columnas, es necesario para ejecutar el programa, organizar dichos datos en una malla para llevar acabo la IBMIU. Los datos de una malla pueden ser guardados en columnas, los datos tienen cierto ciclo de repetición y esta casilla recibe un número entero positivo, que indica cada cuantos hay una repetición de la malla, a continuación se muestra un ejemplo para ser más explícito.

   Vemos los datos del primer archivo .txt que debe ser ingresados

```
 x   y   z
 1  11  0.0909
 1  14  0.0714
 1  17  0.0588
 1  20  0.0500
 4  11  0.3636
 4  14  0.2857
 4  17  0.2353
 4  20  0.2000
 7  11  0.6364
 7  14  0.5000
 7  17  0.4118
 7  20  0.3500
10  11  0.9091
10  14  0.7143
10  17  0.5882
10  20  0.5000
```

   notemos que para la columna x se repite un número por ciclos de cuatro, este es el ciclo que pide la interfaz para poder organizar los datos en matrices y así Matlab pueda construir la malla para poder implementar el método

```
X =

   1     4     7    10
   1     4     7    10
   1     4     7    10
   1     4     7    10

Y =

   11    11    11    11
   14    14    14    14
   17    17    17    17
   20    20    20    20

Z =

   0.0909    0.3636    0.6364    0.9091
   0.0714    0.2857    0.5000    0.7143
   0.0588    0.2353    0.4118    0.5882
   0.0500    0.2000    0.3500    0.5000
```

una vez organizados los datos, Matlab puede graficar la malla.

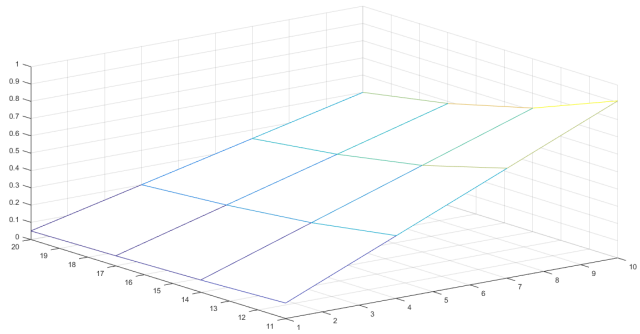


Figura 4.3: Malla

5. Una vez cargados ambos archivos .txt con su información correspondiente y agregando los ciclos de repetición, al presionar el botón interpolar, genera un archivo .txt que contiene los valores aproximados mediante el método en los puntos (x, y).

6. Para graficar y suavizar la superficie, es necesario crear una partición en el eje x para generar puntos que se interpolarán y graficarán, el salto entre ellos se indica en la distancia que habrá entre los nuevos puntos, tomando el ejemplo de la malla anterior, si indicamos el salto entre puntos de 0.5 se generá la nueva malla y se aproximarán los nuevos puntos mediante el método, como se muestra a continuación.

```
X=
1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5
9.0 9.5 10.0
1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5
9.0 9.5 10.0
1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5
9.0 9.5 10.0
1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5
9.0 9.5 10.0
1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5
9.0 9.5 10.0
1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5
9.0 9.5 10.0
1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5
9.0 9.5 10.0
1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5
9.0 9.5 10.0
1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5
9.0 9.5 10.0
1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5
9.0 9.5 10.0
1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5
9.0 9.5 10.0
1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5
9.0 9.5 10.0
1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5
9.0 9.5 10.0
1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5
9.0 9.5 10.0
1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5
9.0 9.5 10.0
1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5
9.0 9.5 10.0
1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5
```

```
9.0 9.5 10.0
1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5
9.0 9.5 10.0
1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5
9.0 9.5 10.0
```

7. De la misma forma que 6 se crea una partición en el eje y creando la nueva matriz para graficar una suavización

```
Y=
11.0 11.0 11.0 11.0 11.0 11.0 11.0 11.0 11.0 11.0 11.0 11.0 11.0
11.0 11.0 11.0 11.0 11.0 11.0
11.5 11.5 11.5 11.5 11.5 11.5 11.5 11.5 11.5 11.5 11.5 11.5 11.5
11.5 11.5 11.5 11.5 11.5 11.5
12.0 12.0 12.0 12.0 12.0 12.0 12.0 12.0 12.0 12.0 12.0 12.0 12.0
12.0 12.0 12.0 12.0 12.0 12.0
12.5 12.5 12.5 12.5 12.5 12.5 12.5 12.5 12.5 12.5 12.5 12.5 12.5
12.5 12.5 12.5 12.5 12.5 12.5
13.0 13.0 13.0 13.0 13.0 13.0 13.0 13.0 13.0 13.0 13.0 13.0 13.0
13.0 13.0 13.0 13.0 13.0 13.0
13.5 13.5 13.5 13.5 13.5 13.5 13.5 13.5 13.5 13.5 13.5 13.5 13.5
13.5 13.5 13.5 13.5 13.5 13.5
14.0 14.0 14.0 14.0 14.0 14.0 14.0 14.0 14.0 14.0 14.0 14.0 14.0
14.0 14.0 14.0 14.0 14.0 14.0
14.5 14.5 14.5 14.5 14.5 14.5 14.5 14.5 14.5 14.5 14.5 14.5 14.5
14.5 14.5 14.5 14.5 14.5 14.5
15.0 15.0 15.0 15.0 15.0 15.0 15.0 15.0 15.0 15.0 15.0 15.0 15.0
15.0 15.0 15.0 15.0 15.0 15.0
15.5 15.5 15.5 15.5 15.5 15.5 15.5 15.5 15.5 15.5 15.5 15.5 15.5
15.5 15.5 15.5 15.5 15.5 15.5
16.0 16.0 16.0 16.0 16.0 16.0 16.0 16.0 16.0 16.0 16.0 16.0 16.0
16.0 16.0 16.0 16.0 16.0 16.0
16.5 16.5 16.5 16.5 16.5 16.5 16.5 16.5 16.5 16.5 16.5 16.5 16.5
16.5 16.5 16.5 16.5 16.5 16.5
17.0 17.0 17.0 17.0 17.0 17.0 17.0 17.0 17.0 17.0 17.0 17.0 17.0
17.0 17.0 17.0 17.0 17.0 17.0
17.5 17.5 17.5 17.5 17.5 17.5 17.5 17.5 17.5 17.5 17.5 17.5 17.5
17.5 17.5 17.5 17.5 17.5 17.5
```

```
18.0 18.0 18.0 18.0 18.0 18.0 18.0 18.0 18.0 18.0 18.0 18.0 18.0
18.0 18.0 18.0 18.0 18.0 18.0
18.5 18.5 18.5 18.5 18.5 18.5 18.5 18.5 18.5 18.5 18.5 18.5 18.5
18.5 18.5 18.5 18.5 18.5 18.5
19.0 19.0 19.0 19.0 19.0 19.0 19.0 19.0 19.0 19.0 19.0 19.0 19.0
19.0 19.0 19.0 19.0 19.0 19.0
19.5 19.5 19.5 19.5 19.5 19.5 19.5 19.5 19.5 19.5 19.5 19.5 19.5
19.5 19.5 19.5 19.5 19.5 19.5
20.0 20.0 20.0 20.0 20.0 20.0 20.0 20.0 20.0 20.0 20.0 20.0 20.0
20.0 20.0 20.0 20.0 20.0 20.0

Z=
0.09 0.14 0.18 0.23 0.27 0.32 0.36 0.41 0.45 0.50 0.55 0.59 0.64
0.68 0.73 0.77 0.82 0.86 0.91
0.09 0.13 0.17 0.22 0.26 0.30 0.35 0.39 0.43 0.48 0.52 0.57 0.61
0.65 0.70 0.74 0.78 0.83 0.87
0.08 0.13 0.17 0.21 0.25 0.29 0.33 0.38 0.42 0.46 0.50 0.54 0.58
0.63 0.67 0.71 0.75 0.79 0.83
0.08 0.12 0.16 0.20 0.24 0.28 0.32 0.36 0.40 0.44 0.48 0.52 0.56
0.60 0.64 0.68 0.72 0.76 0.80
0.08 0.12 0.15 0.19 0.23 0.27 0.31 0.35 0.38 0.42 0.46 0.50 0.54
0.58 0.62 0.65 0.69 0.73 0.77
0.07 0.11 0.15 0.19 0.22 0.26 0.30 0.33 0.37 0.41 0.44 0.48 0.52
0.56 0.59 0.63 0.67 0.70 0.74
0.07 0.11 0.14 0.18 0.21 0.25 0.29 0.32 0.36 0.39 0.43 0.46 0.50
0.54 0.57 0.61 0.64 0.68 0.71
0.07 0.10 0.14 0.17 0.21 0.24 0.28 0.31 0.34 0.38 0.41 0.45 0.48
0.52 0.55 0.59 0.62 0.66 0.69
0.07 0.10 0.13 0.17 0.20 0.23 0.27 0.30 0.33 0.37 0.40 0.43 0.47
0.50 0.53 0.57 0.60 0.63 0.67
0.06 0.10 0.13 0.16 0.19 0.23 0.26 0.29 0.32 0.35 0.39 0.42 0.45
0.48 0.52 0.55 0.58 0.61 0.65
0.06 0.09 0.13 0.16 0.19 0.22 0.25 0.28 0.31 0.34 0.38 0.41 0.44
0.47 0.50 0.53 0.56 0.59 0.63
0.06 0.09 0.12 0.15 0.18 0.21 0.24 0.27 0.30 0.33 0.36 0.39 0.42
0.45 0.48 0.52 0.55 0.58 0.61
0.06 0.09 0.12 0.15 0.18 0.21 0.24 0.26 0.29 0.32 0.35 0.38 0.41
0.44 0.47 0.50 0.53 0.56 0.59
0.06 0.09 0.11 0.14 0.17 0.20 0.23 0.26 0.29 0.31 0.34 0.37 0.40
0.43 0.46 0.49 0.51 0.54 0.57
0.06 0.08 0.11 0.14 0.17 0.19 0.22 0.25 0.28 0.31 0.33 0.36 0.39
```

```
0.42 0.44 0.47 0.50 0.53 0.56
0.05 0.08 0.11 0.14 0.16 0.19 0.22 0.24 0.27 0.30 0.32 0.35 0.38
0.41 0.43 0.46 0.49 0.51 0.54
0.05 0.08 0.11 0.13 0.16 0.18 0.21 0.24 0.26 0.29 0.32 0.34 0.37
0.39 0.42 0.45 0.47 0.50 0.53
0.05 0.08 0.10 0.13 0.15 0.18 0.21 0.23 0.26 0.28 0.31 0.33 0.36
0.38 0.41 0.44 0.46 0.49 0.51
0.05 0.08 0.10 0.13 0.15 0.18 0.20 0.23 0.25 0.28 0.30 0.33 0.35
0.38 0.40 0.43 0.45 0.48 0.50
```

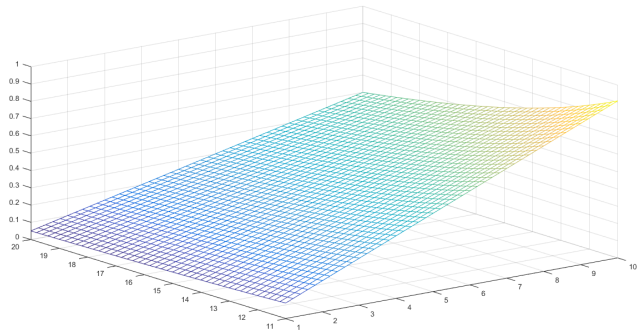


Figura 4.4: Suavización de la malla

8. Una vez que es cargada la información requerida, el botón grafica una aproximación de la superficie, se muestra la gráfica estimada mediante el método , además una nueva ventana aparece con la gráfica suavizada y con los puntos estimados de (x, y) y su ubicación en la superficie.

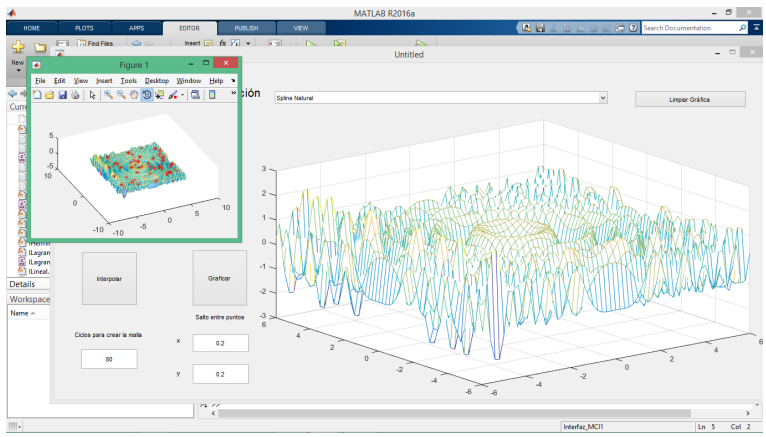


Figura 4.5: Gráfica en la interfaz

9. Ventana donde aparece la gráfica de la interpolación.

10. Botón que limpia la gráfica después de un proceso.

4.2.2. Interfaz para Interpolación Bivariada de Akima

La imagen de la interfaz para el método de Interpolación Bivariada de Akima es la siguiente

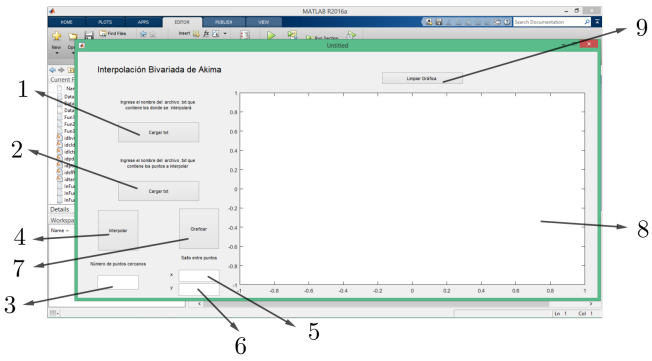


Figura 4.6: Interfaz de la Interpolación Bivariada de Akima

1. Documentos .txt con las características de 1 de la interfaz para IBMIU.

2. Documentos .txt con las características de 2 de la interfaz para IBMIU.

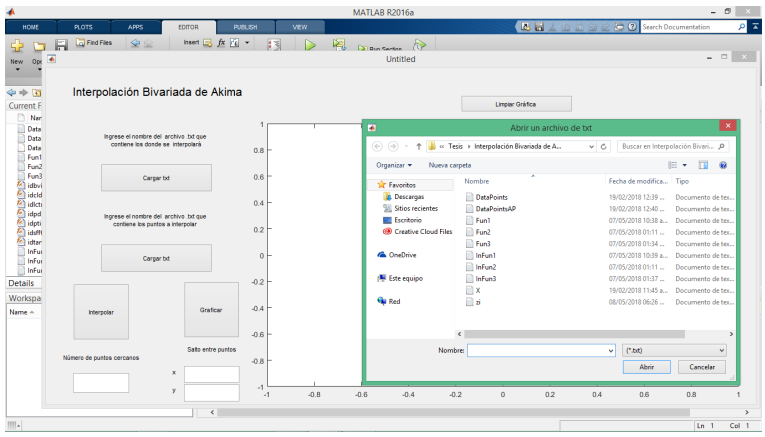


Figura 4.7: Interfaz de la Interpolación Bivariada de Akima

3. Ingresa el valor entero positivo del número de puntos cercanos que se considerarán para estimar las derivadas parciales del método, debe ser mayor a tres.

4. Realiza la misma función que 5 de la interfaz IBMIU.
5. La misma función que 6 de la interfaz IBMIU.
6. La misma función que 7 de la interfaz IBMIU.
7. La misma función que 8 de la interfaz IBMIU

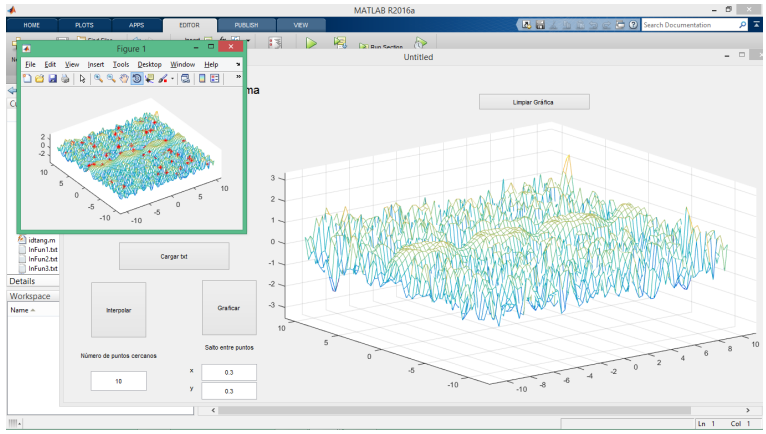


Figura 4.8: Gráfica de la Interfaz

8. Ventana donde aparece la gráfica de la interpolación.
9. Botón que limpia la gráfica después de un proceso.

Es importante mencionar, el costo computacional de los métodos, por una lado el tiempo de ejecución del interpolante IBMIU es mejor que el Interpolante Bivariado de Akima (se analizará más a detalle en el siguiente capítulo), ahora una vez que tenemos los programas implementados, es necesario analizar su comportamientos en los datos y concluir que tipo de resultados ofrece.

Capítulo 5

Conclusiones

En los capítulos dos y tres estudiamos la teoría en los métodos de interpolación, ahora veremos algunos resultados obtenidos, una vez implementados mediante programas en el software numérico Matlab en su versión 2016, primero los aplicaremos a algunas funciones analíticas para comparar el resultados reales y estimados por los interpolantes, después usaremos para poder estimar los resultados sobre las superficies que se usan como herramienta de cálculo de propiedades de los fluidos en yacimientos de petróleos.

En el caso de la IBMIU usaremos los siguientes interpolantes para implementarlo

1. Spline Cúbico Natural.

2. Spline Not-A-Knot.

3. Interpolación Lineal.

4. Interpolación Cúbica de Akima

5. Interpolación Restringida.

6. Interpolación de Preservación. Monótona

Nota: la interpolación de Lagrange, Hermite y Newton no se usarán debido a que el grado de los polinomios es grande con lo cual el polinomio tiene curvas muy pronunciadas, como consecuencia la distancia entre el valor real y estimado es muy grande, lo cual resulta ineficiente, en el caso del Spline Cúbico Completo, es necesario conocer el valor de las derivadas en los puntos frontera, lo cual no siempre es posible tener dentro de los datos originales.

5.1. Interpolación Univariada

Antes de analizar los interpolantes bidimensionales, analicemos su comportamiento en forma univariada para conocer sus características, para hacerlo, utilizaremos las siguiente lista de puntos.

Cuadro 5.1: Cuadro con datos

x	y
10.8724	6.6347
13.7967	6.8361
21.1833	2.7942
37.4934	3.1524
43.6586	6.3842
71.7471	4.8199
74.6233	5.9897
78.6149	2.8916
79.6756	2.4170
85.0460	7.6371
92.0875	7.1354

5.1.1. Interpolantes Polinomiales

A continuación se muestran los datos interpolados mediante los métodos de Newton y Hermite

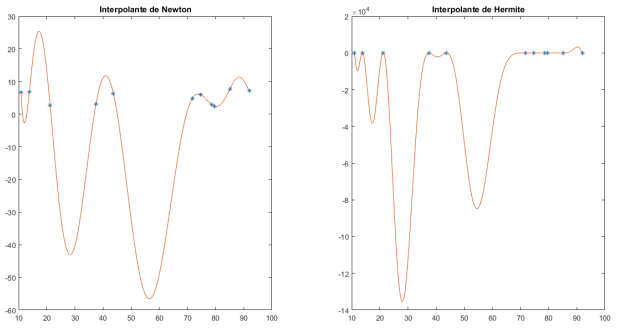


Figura 5.1

en la imagen podemos observar como en ambos métodos la escala de los polinomios rebasa por mucho a la de los datos del Cuadro 5.1, por lo cual sen tendría errores muy grandes al utilizarlos en el interpolante IBMIU.

5.1.2. Interpolantes Polinomiales a Trozos

Debido a los grandes que errores que generan los interpolantes polinomiales, analicemos el comportamientos de los interpolantes por intervalos (Spline Cúbico y Hermite Cúbico) como alternativa para reducir el error

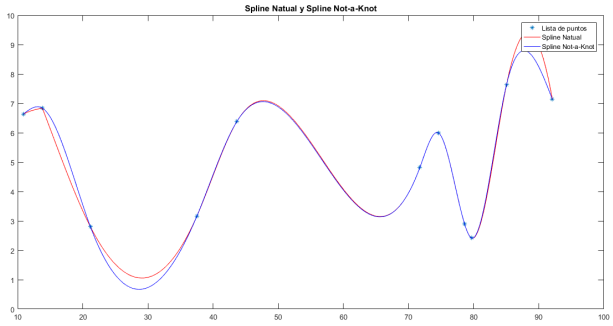


Figura 5.2

en la figura 5.2 anterior observamos la forma de interpolar de ambos Spline, donde notamos algunas diferencias cerca de la frontera

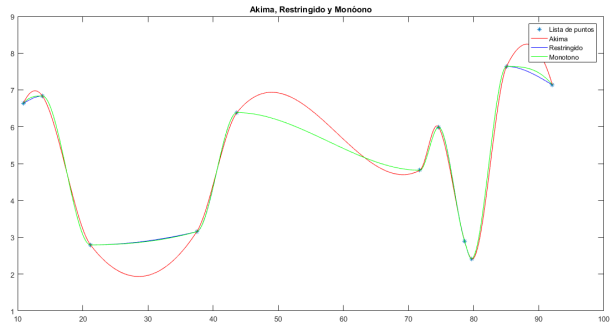


Figura 5.3

en los interpolates Hermite Cúbico de la figura 5.4 (Akima, Restringido y Monótono) observemos por las curvas generadas que el interpolante de Akima suaviza de mejor manera que los otros dos, generando algunas curvas sin ser muy pronunciadas, ahora comparemos la interpolación de Akima con el interpolante Spline vista en la figura 5.4

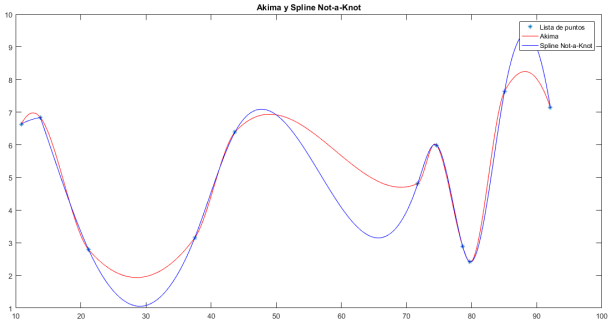
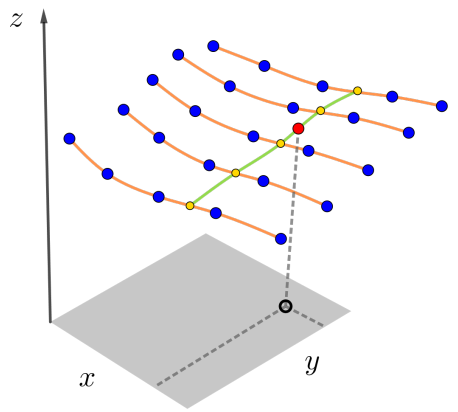


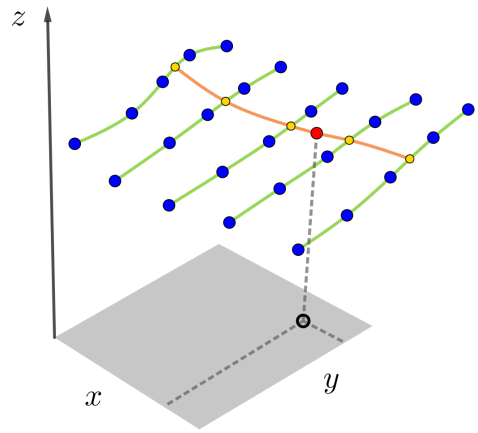
Figura 5.4

podemos notar que el interpolante de Akima realiza una suavización con curvas menos pronunciadas que el Spline, este resultado será importante para analizar el comportamiento de los interpolantes IBMIU, ya que nos permite comprender mejor los resultados obtenidos en interpolación bivariada.

5.2. Funciones Analíticas

Antes de analizar la eficiencia de los interpolates sobre funciones analíticas, analizaremos si existen diferencias numéricas significativas al cambiar el orden de interpolación de los IBMIU, es decir, si existen diferencias importantes al interpolar primero sobre el plano (x, z) y después sobre el plano (y, z) y comparar con los resultados obtenidos al interpolar sobre el plano (y, z) y posteriormente sobre el plano (x, z) como se muestra en las siguientes imágenes





para compararlos se realizaron pruebas en mil datos interpolados, obteniendo los siguientes resultados, que representan la diferencia en valor absolutos de ambas formas de interpolar

Intervalo		*Frecuencia*
0	$7.105427357601E-15$	954
$7.105427357601E-15$	$1.4210854715202E-14$	33
$1.42108547152E-14$	$2.1316282072803E-14$	11
$2.13162820728E-14$	$2.8421709430404E-14$	2

se puede notar que la diferencia numérica al invertir el orden de interpolación es muy pequeña para todos los casos por lo cual el orden de interpolación de los IBMIU no representa un aspecto significativo, en nuestro caso siempre lo haremos en el orden de los planos (x, z) y (y, z). Finalmente mostramos quince

datos obtenidos con ambas formas de interpolación.

Interpolante		*Diferencia*
$(x,z) \longrightarrow (y,z)$	$(y,z) \longrightarrow (x,z)$	$\lvert(x,z) \longrightarrow (y,z) - (y,z) \longrightarrow (x,z)\rvert$
23.2082683189647	23.2082683189647	0.00000000000000000000
−12.625617246642	−12.625617246642	0.00000000000000000000
64.7802371233969	64.7802371233969	0.00000000000001421085
−13.5018572218662	−13.5018572218662	0.00000000000000177636
40.4252060404357	40.4252060404357	0.00000000000000000000
63.6044154402264	63.6044154402264	0.00000000000000000000
−4.54651383275162	−4.54651383275162	0.00000000000000088818
8.09322608102197	8.09322608102197	0.00000000000000000000
1.25444453364654	1.25444453364654	0.00000000000000044409
−1.73472931601773	−1.73472931601773	0.00000000000000022204
−12.2160628619673	−12.2160628619673	0.00000000000000000000
85.0452791065428	85.0452791065428	0.00000000000000000000
56.7701559930283	56.7701559930283	0.00000000000000000000

A continuación estimaremos puntos con los interpolantes en algunas funciones, también usaremos el cálculo del error absoluto (Δz) y relativo ε_r para comparar los valores originales $(z_o = f(x,y))$ y los estimados z_e

$$\Delta z = |z_e - z_o|$$

$$\varepsilon_r = \left| \frac{\Delta z}{z_o} \right|$$

vemos la gráfica de la función $\sin(\sin(xy) + \cos(xy))$

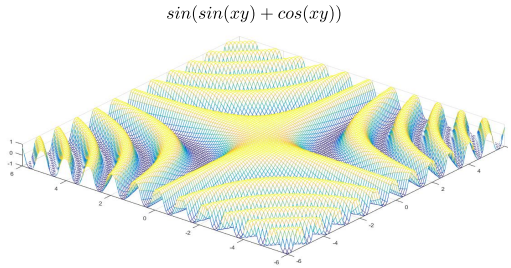


Figura 5.5

en la siguiente tabla se muestran treinta datos tomados de cinco mil estimados con IBMIU con cada interpolante y la Interpolación Bivariada de Akima

Cuadro 5.2: Estimación de treinta puntos (x_{ii}, y_{ii}) en la función $z_o = f(x, y) = \sin(\sin(xy) + \cos(xy))$

x_{ii}	y_{ii}	z_o	S.Nat. (z_e)	Δz	ε_r
1.83037	1.40514	-0.29812	-0.29794	0.00019	0.00062
-1.41044	4.88863	0.24166	0.24249	0.00083	0.00341
1.38884	-0.34428	0.41478	0.41478	0.00000	0.00001
-4.65912	2.41272	0.93671	0.96887	0.03217	0.03434
1.30741	0.53204	0.98686	0.98681	0.00005	0.00005
-2.76124	-3.16486	-0.14010	-0.14086	0.00075	0.00538
-3.50024	0.57373	-0.97102	-0.97203	0.00101	0.00104
-3.29371	0.84090	-0.96222	-0.97963	0.01742	0.01810
0.70173	4.28751	-0.75698	-0.69622	0.06076	0.08027
0.73836	-2.43522	-0.93190	-0.92998	0.00191	0.00205
1.72857	4.80360	0.44962	0.44115	0.00847	0.01884
3.41861	-0.53803	-0.94231	-0.99529	0.05298	0.05623
1.91158	3.65200	0.98691	0.99576	0.00885	0.00897
4.11965	4.53629	0.73525	0.58679	0.14846	0.20191
-2.40219	4.91606	0.98768	0.95723	0.03045	0.03083
-0.44314	-3.52499	0.84613	0.84613	0.00000	0.00000
-4.05139	2.29767	-0.89527	-0.88049	0.01478	0.01650
-4.36308	-1.61046	0.98757	0.99376	0.00619	0.00627
3.63112	-1.84284	0.49742	0.49574	0.00168	0.00338
-1.89679	1.25731	-0.98768	-0.98799	0.00032	0.00032
4.61502	2.99518	0.95215	0.95694	0.00479	0.00503
-1.82210	3.82679	0.13486	0.13509	0.00022	0.00167
-1.33935	2.84690	-0.16016	-0.15997	0.00019	0.00118
0.29577	-2.28851	0.15257	0.15214	0.00043	0.00283
-2.33183	0.49028	-0.47535	-0.47523	0.00012	0.00025
1.52536	-1.24328	-0.95432	-0.95433	0.00001	0.00001
1.92777	-2.23809	0.50934	0.49016	0.01918	0.03765
4.38172	2.17506	-0.89119	-0.80692	0.08426	0.09455
1.70538	-4.81989	-0.96125	-0.93838	0.02288	0.02380
-3.40529	1.01238	-0.60716	-0.60733	0.00017	0.00028

S.Not. (z_e)	Δz	ε_r	ILin. (z_e)	Δz	ε_r
-0.29794	0.00019	0.00062	-0.28953	0.00859	0.02883
0.24249	0.00083	0.00341	0.23949	0.00218	0.00902
0.41478	0.00000	0.00001	0.41368	0.00110	0.00265
0.96887	0.03217	0.03434	0.76250	0.17420	0.18598
0.98681	0.00005	0.00005	0.98581	0.00105	0.00107
-0.14086	0.00075	0.00538	-0.11905	0.02105	0.15025
-0.97203	0.00101	0.00104	-0.94919	0.02182	0.02247
-0.97963	0.01742	0.01810	-0.85080	0.11142	0.11579
-0.69622	0.06076	0.08027	-0.49714	0.25984	0.34326
-0.92998	0.00191	0.00205	-0.86050	0.07139	0.07661
0.44115	0.00847	0.01884	0.35703	0.09260	0.20594
-0.99529	0.05298	0.05623	-0.77166	0.17065	0.18110
0.99576	0.00885	0.00897	0.95215	0.03476	0.03522
0.58679	0.14846	0.20191	-0.14066	0.87591	1.19131
0.95723	0.03045	0.03083	0.62649	0.36118	0.36569
0.84613	0.00000	0.00000	0.84144	0.00470	0.00555
-0.88049	0.01478	0.01650	-0.72626	0.16900	0.18877
0.99376	0.00619	0.00627	0.93551	0.05206	0.05271
0.49574	0.00168	0.00338	0.45747	0.03995	0.08032
-0.98799	0.00032	0.00032	-0.98409	0.00358	0.00363
0.95694	0.00479	0.00503	0.90131	0.05085	0.05340
0.13509	0.00022	0.00167	0.12779	0.00707	0.05245
-0.15997	0.00019	0.00118	-0.15507	0.00509	0.03178
0.15214	0.00043	0.00283	0.14147	0.01110	0.07277
-0.47523	0.00012	0.00025	-0.46380	0.01155	0.02430
-0.95433	0.00001	0.00001	-0.95398	0.00034	0.00035
0.49016	0.01918	0.03765	0.37241	0.13692	0.26883
-0.80692	0.08426	0.09455	-0.54444	0.34675	0.38908
-0.93838	0.02288	0.02380	-0.63905	0.32220	0.33519
-0.60733	0.00017	0.00028	-0.60441	0.00275	0.00453

Ak. (z_e)	Δz	ε_r	I.R. (z_e)	Δz	ε_r
-0.29512	0.00301	0.01008	-0.29437	0.00375	0.01259
0.24617	0.00451	0.01866	0.24667	0.00500	0.02070
0.41408	0.00070	0.00170	0.41552	0.00074	0.00178
0.92314	0.01357	0.01449	0.81125	0.12545	0.13393
0.98751	0.00065	0.00066	0.98634	0.00052	0.00053
-0.12932	0.01078	0.07698	-0.17369	0.03359	0.23975
-0.98389	0.01287	0.01326	-0.96819	0.00283	0.00291
-0.93410	0.02812	0.02922	-0.89462	0.06759	0.07025
-0.70156	0.05542	0.07321	-0.65385	0.10313	0.13624
-0.92909	0.00281	0.00301	-0.91121	0.02068	0.02219
0.40369	0.04593	0.10216	0.39297	0.05666	0.12601
-0.94314	0.00084	0.00089	-0.78173	0.16057	0.17041
0.96937	0.01754	0.01777	0.95849	0.02842	0.02879
0.40682	0.32842	0.44668	-0.03277	0.76802	1.04457
0.77795	0.20972	0.21234	0.80961	0.17807	0.18029
0.84676	0.00063	0.00074	0.84553	0.00060	0.00071
-0.86250	0.03276	0.03660	-0.82704	0.06823	0.07621
0.98085	0.00672	0.00680	0.97086	0.01671	0.01692
0.49048	0.00694	0.01395	0.48321	0.01421	0.02856
-0.98855	0.00087	0.00088	-0.98495	0.00273	0.00276
0.95220	0.00005	0.00005	0.92323	0.02892	0.03037
0.13165	0.00321	0.02381	0.13343	0.00144	0.01066
-0.15583	0.00432	0.02700	-0.15979	0.00037	0.00231
0.15008	0.00249	0.01634	0.14772	0.00485	0.03179
-0.47685	0.00150	0.00316	-0.47099	0.00436	0.00916
-0.95428	0.00004	0.00005	-0.95443	0.00011	0.00011
0.46101	0.04833	0.09488	0.44822	0.06112	0.11999
-0.75756	0.13363	0.14995	-0.73951	0.15168	0.17020
-0.89456	0.06670	0.06939	-0.79157	0.16968	0.17652
-0.61272	0.00556	0.00916	-0.61358	0.00642	0.01058

IMon. (z_e)	Δz	ε_r	Biv.Ak. (z_e)	Δz	ε_r
-0.29484	0.00329	0.01102	-0.30233	0.00420	0.01391
0.24732	0.00565	0.02338	0.22479	0.01688	0.07509
0.41536	0.00058	0.00139	0.41508	0.00030	0.00071
0.82653	0.11017	0.11762	0.92607	0.01064	0.01149
0.98632	0.00054	0.00055	0.98544	0.00142	0.00144
-0.17127	0.03116	0.22244	-0.13649	0.00361	0.02645
-0.96798	0.00304	0.00313	-0.97973	0.00871	0.00889
-0.90509	0.05713	0.05937	-0.96455	0.00233	0.00242
-0.67067	0.08631	0.11402	-0.76106	0.00408	0.00536
-0.91736	0.01453	0.01559	-0.93256	0.00066	0.00071
0.40059	0.04903	0.10905	0.47186	0.02224	0.04713
-0.78543	0.15688	0.16649	-0.94002	0.00229	0.00243
0.95983	0.02708	0.02743	0.99071	0.00380	0.00384
0.01374	0.72151	0.98131	0.69836	0.03689	0.05282
0.81244	0.17524	0.17743	1.00489	0.01721	0.01713
0.84560	0.00053	0.00063	0.85699	0.01086	0.01267
-0.84198	0.05328	0.05952	-0.92565	0.03038	0.03282
0.97301	0.01457	0.01475	1.07545	0.08788	0.08172
0.48595	0.01147	0.02306	0.50373	0.00632	0.01254
-0.98498	0.00270	0.00273	-0.98839	0.00072	0.00073
0.92459	0.02756	0.02894	0.94388	0.00827	0.00876
0.13351	0.00136	0.01005	0.14371	0.00885	0.06156
-0.16029	0.00013	0.00081	-0.17697	0.01682	0.09502
0.14872	0.00385	0.02523	0.15605	0.00348	0.02230
-0.47194	0.00341	0.00717	-0.47838	0.00304	0.00635
-0.95434	0.00002	0.00002	-0.95108	0.00324	0.00341
0.45972	0.04962	0.09743	0.49695	0.01239	0.02493
-0.76402	0.12717	0.14270	-0.88781	0.00337	0.00380
-0.81288	0.14838	0.15436	-0.97597	0.01471	0.01508
-0.61214	0.00498	0.00821	-0.60368	0.00348	0.00576

5.2.1. Análisis de Datos

A continuación se presentan algunos resultados encontrados con cinco mil puntos interpolados, se muestran una tabla con promedios de Δz y ε_r e histogramas para cada interpolante

Cuadro 5.3: Promedios de Δ_z y ε_r

Interpolante	Promedio Δz	Promedio ε_r
Spline Natural	0.02705	16.327 %
Spline Not-a-Knot	0.02705	16.327 %
Lineal	0.10409	39.249 %
Akima	0.04726	25.799 %
Restringido	0.06519	30.178 %
Monónoto	0.06144	28.899 %
Bivariado de Akima	0.01832	7.2477 %

los siguientes histogramas muestran la frecuencia del error tanto absoluto como relativo para cada interpolante

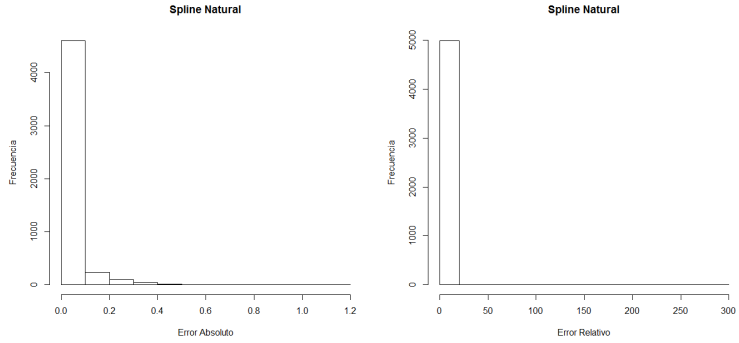


Figura 5.6

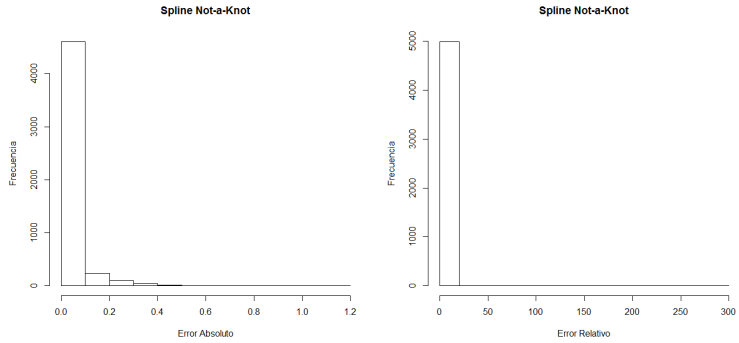


Figura 5.7

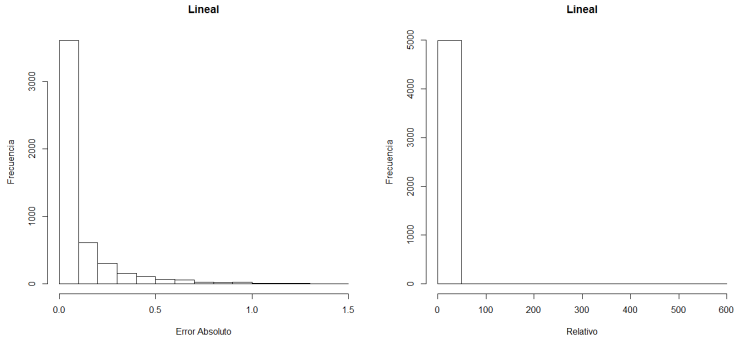


Figura 5.8

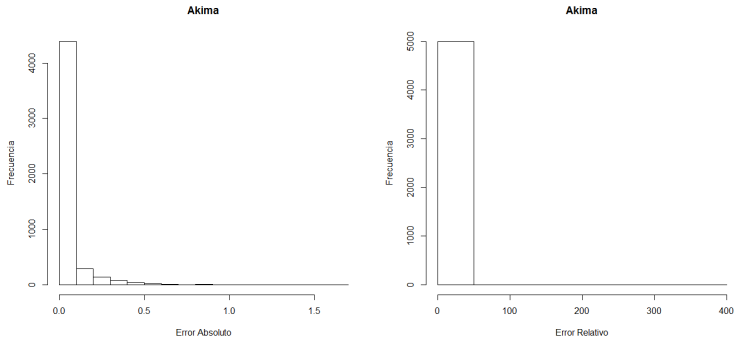


Figura 5.9

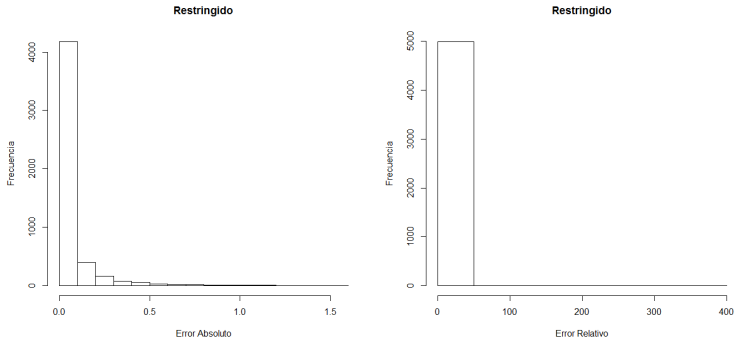


Figura 5.10

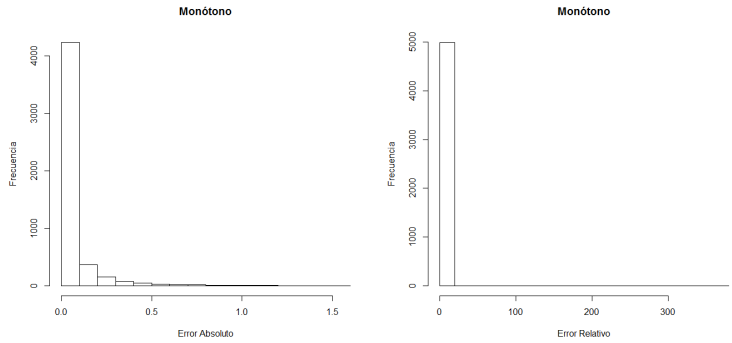


Figura 5.11

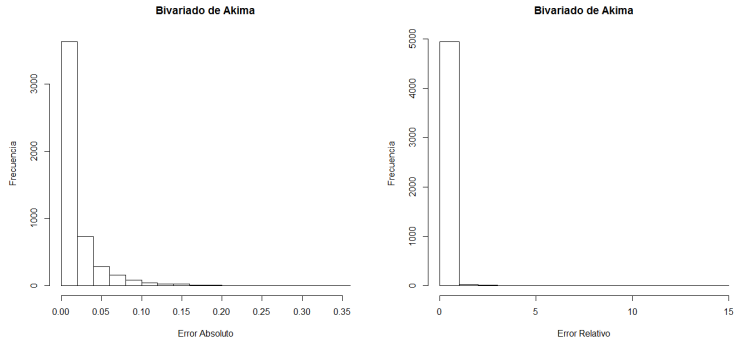


Figura 5.12

Las gráficas del lado izquierdo nos muestran la diferencia que hay entre el valor real de la función analítica y el valor estimado de cinco mil puntos mediante el interpolante, mientras que las gráficas del lado izquierdo nos muestran el grado de error en porcentaje de la misma cantidad de datos, es importante notar que los interpolantes IBMIU Spline y Bivariado de Akima, tiene una mejor disminución del error en funciones analíticas elegidas, esto se puede ver más claro en las siguientes tablas.

Cuadro 5.4: Frecuencias de Δz y ε_r (Spline Natural)

Δz		Datos	ε_r		Datos
0	0.1	4605	0 %	20 %	4994
0.1	0.2	231	20 %	40 %	3
0.2	0.3	95	60 %	80 %	2
0.3	0.4	40	280 %	300 %	1
0.4	0.5	13			
0.5	0.6	4			
0.6	0.7	4			
0.7	0.8	1			
0.8	0.9	2			
0.9	1	2			
1	1.1	2			
1.1	1.2	1			

Cuadro 5.5: Frecuencias de Δz y ε_r (Spline Not-a-Knot)

Δz		Datos	ε_r		Datos
0	0.1	4605	0 %	20 %	4994
0.1	0.2	231	20 %	40 %	3
0.2	0.3	95	60 %	80 %	2
0.3	0.4	40	280 %	300 %	1
0.4	0.5	13			
0.5	0.6	4			
0.6	0.7	4			
0.7	0.8	1			
0.8	0.9	2			
0.9	1	2			
1	1.1	2			
1.1	1.2	1			

Cuadro 5.6: Frecuencias de Δz y ε_r (Lineal)

Δz		Datos	ε_r		Datos
0	0.1	3607	0 %	50 %	4996
0.1	0.2	611	50 %	100 %	2
0.2	0.3	303	100 %	150 %	1
0.3	0.4	159	550 %	600 %	1
0.4	0.5	105			
0.5	0.6	66			
0.6	0.7	55			
0.7	0.8	24			
0.8	0.9	17			
0.9	1	24			
1	1.1	10			
1.1	1.2	8			
1.2	1.3	7			
1.3	1.4	3			
1.4	1.5	1			

Cuadro 5.7: Frecuencias de Δz y ε_r (Akima)

Δz		Datos	ε_r		Datos
0	0.1	4388	0 %	50 %	4998
0.1	0.2	289	50 %	100 %	1
0.2	0.3	137	350 %	400 %	1
0.3	0.4	79			
0.4	0.5	42			
0.5	0.6	25			
0.6	0.7	11			
0.7	0.8	4			
0.8	0.9	9			
0.9	1	2			
1	1.1	3			
1.1	1.2	3			
1.2	1.3	2			
1.3	1.4	0			
1.4	1.5	2			
1.5	1.6	3			
1.6	1.7	1			

Cuadro 5.8: Frecuencias de Δz y ε_r (Restringido)

Δz		Datos	ε_r		Datos
0	0.1	4179	0 %	50 %	4997
0.1	0.2	401	50 %	100 %	1
0.2	0.3	159	100 %	150 %	1
0.3	0.4	78	350 %	400 %	1
0.4	0.5	55			
0.5	0.6	31			
0.6	0.7	22			
0.7	0.8	23			
0.8	0.9	13			
0.9	1	12			
1	1.1	8			
1.1	1.2	9			
1.2	1.3	4			
1.3	1.4	3			
1.4	1.5	2			
1.5	1.6	1			

Cuadro 5.9: Frecuencias de Δz y ε_r (Monótono)

Δz		Datos	ε_r		Datos
0	0.1	4236	0.00 %	20.00 %	4993
0.1	0.2	365	20.00 %	40.00 %	4
0.2	0.3	151	60.00 %	80.00 %	1
0.3	0.4	78	340.00 %	380.00 %	1
0.4	0.5	46			
0.5	0.6	31			
0.6	0.7	20			
0.7	0.8	24			
0.8	0.9	11			
0.9	1	12			
1	1.1	10			
1.1	1.2	6			
1.2	1.3	3			
1.3	1.4	4			
1.4	1.5	2			
1.5	1.6	1			

Cuadro 5.10: Frecuencias de Δz y ε_r (Bivariado de Akima)

Δz		Datos	ε_r		Datos
0	0.02	3630	0 %	1 %	4951
0.02	0.04	731	1 %	2 %	19
0.04	0.06	279	2 %	3 %	16
0.06	0.08	155	3 %	4 %	5
0.08	0.1	87	4 %	5 %	2
0.1	0.12	42	5 %	6 %	2
0.12	0.14	27	6 %	7 %	1
0.14	0.16	27	7 %	8 %	1
0.16	0.18	10	8 %	9 %	1
0.18	0.2	5	9 %	10 %	0
0.2	0.22	3	10 %	11 %	0
0.22	0.24	1	11 %	12 %	1
0.24	0.26	1	12 %	13 %	0
0.26	0.28	0	13 %	14 %	0
0.28	0.3	0	14 %	15 %	1
0.3	0.32	1			
0.32	0.34	0			
0.34	0.36	1			

En los datos de las tablas podemos confirmar que la mejor disminución del error absoluto y relativo en funciones analíticas es mediante el interpolante IBMIU Spline y Bivariado de Akima, sin embargo es necesario mencionar que los resultados obtenidos provienen de funciones con secciones “suaves”, ahora analizaremos mediante gráficas el comportamiento de los interpolantes en las secciones de las funciones donde no tienen las característica de ser suaves.

5.2.2. Gráficas Generadas por los Interpolantes

En esta sección se mostrarán gráficas de algunas funciones, se interpolarán mediante los métodos mencionados previamente, las funciones tienen la características de ser "suaves" en algunas secciones, mientras que en otras no tiene dicha característica.

1. $f(x, y) = \sin(x^2 + y^2)$

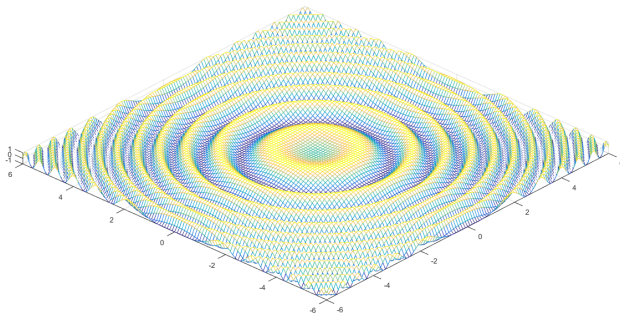
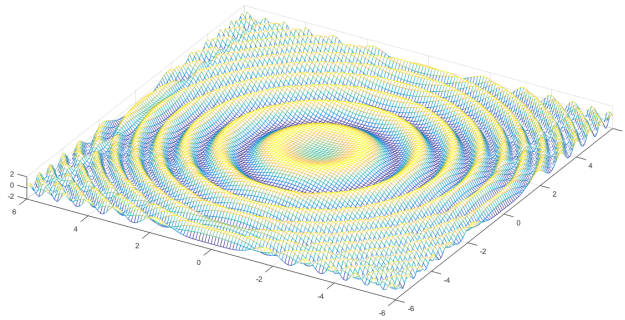
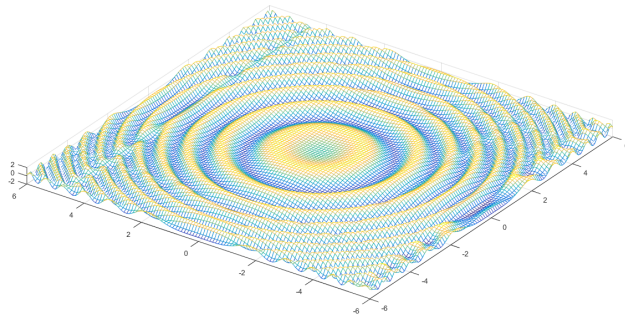


Figura 5.13: $\sin(x^2 + y^2)$

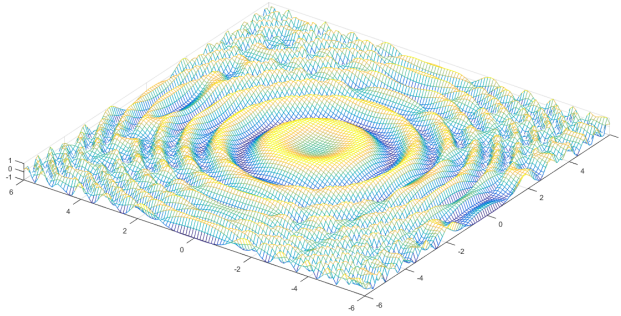
a) Spline Natural



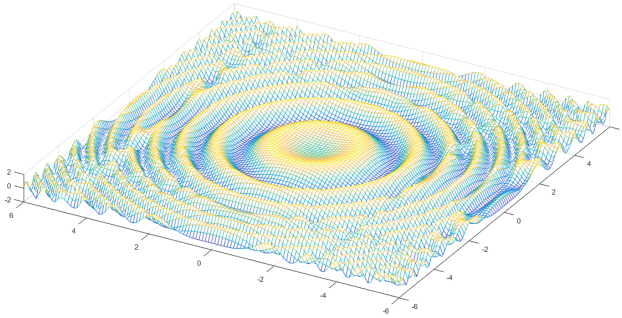
b) Spline Not-A-Knot



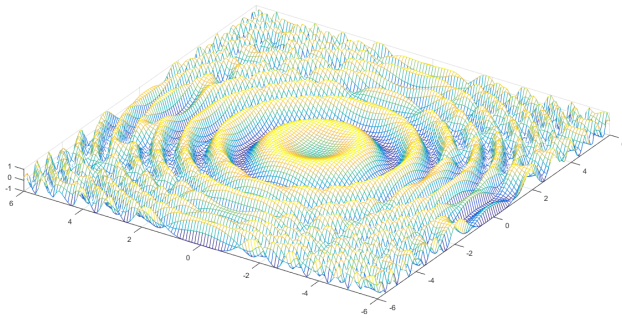
c) Lineal



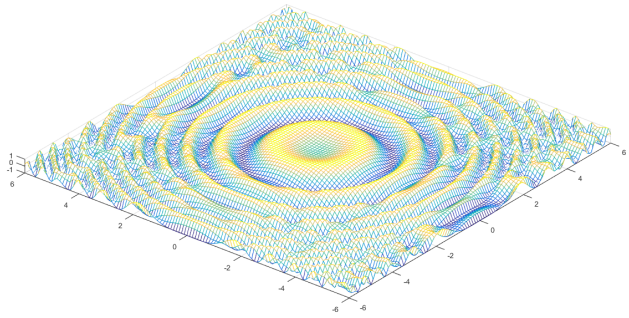
d) Akima



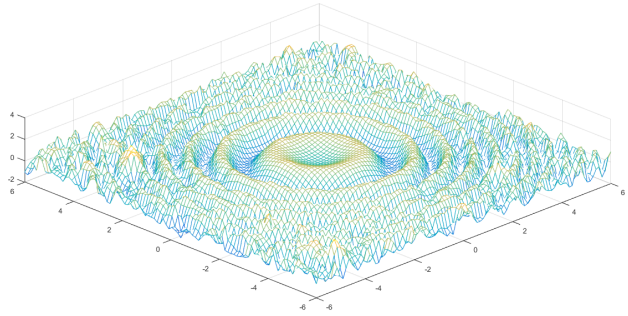
e) Restringido



f) Monótono



g) Bivariado de Akima



2. $\cos(y^2 + \sin(x))$

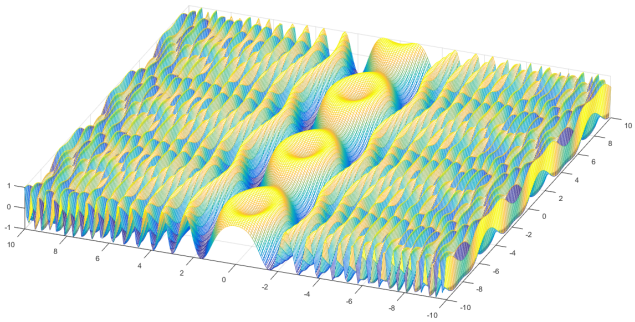
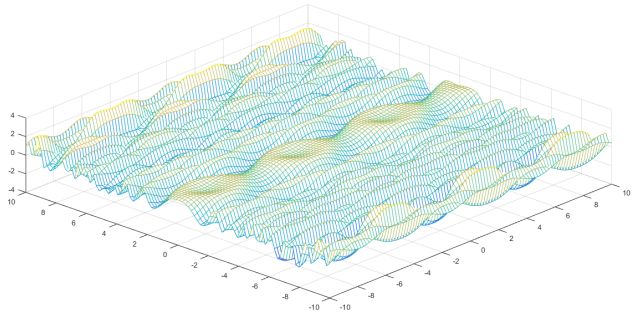
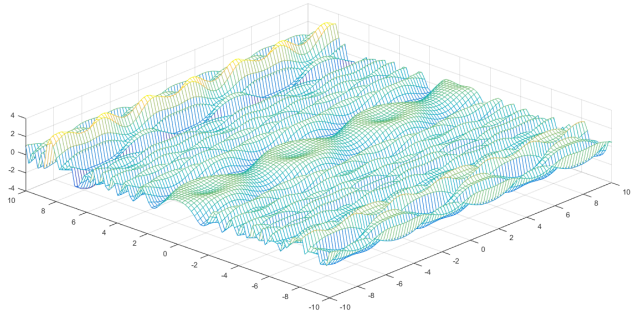


Figura 5.14: $\cos(y^2 + \sin(x))$

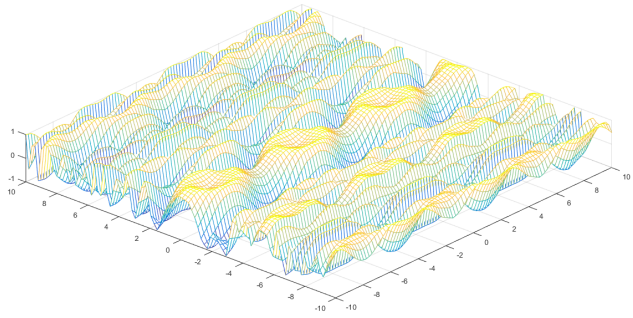
a) Spline Natural



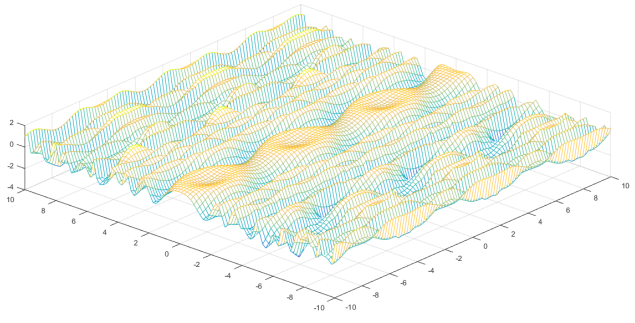
b) Spline Not-A-Knot



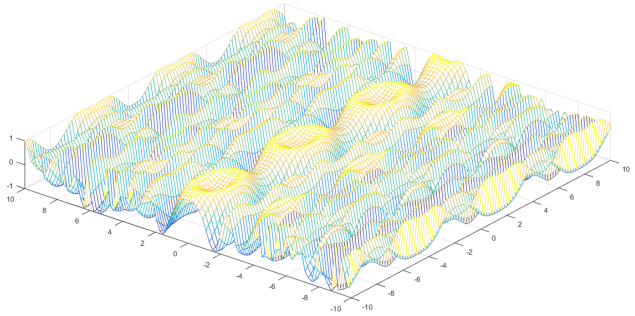
c) Lineal



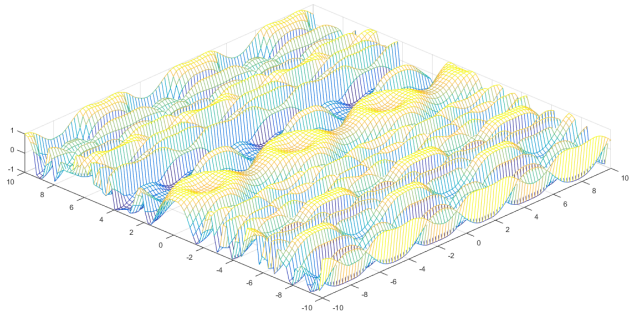
d) Akima



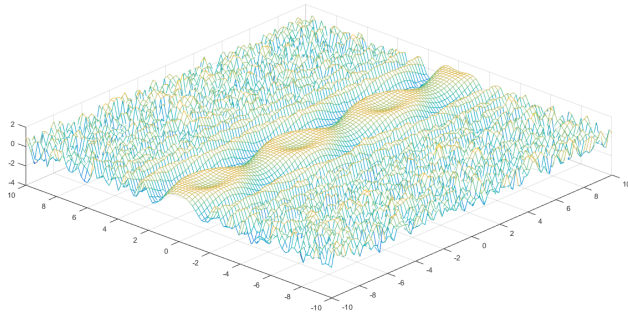
e) Restringido



f) Monótono



g) Bivariado de Akima



3. $f(x, y) = e^{sin(x)+cos(y)}$

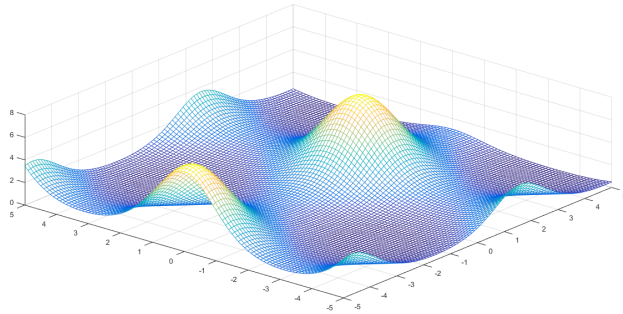
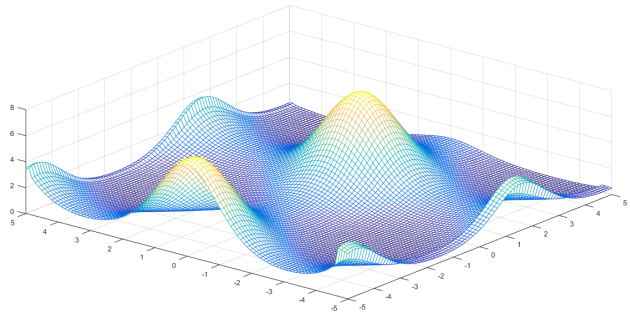
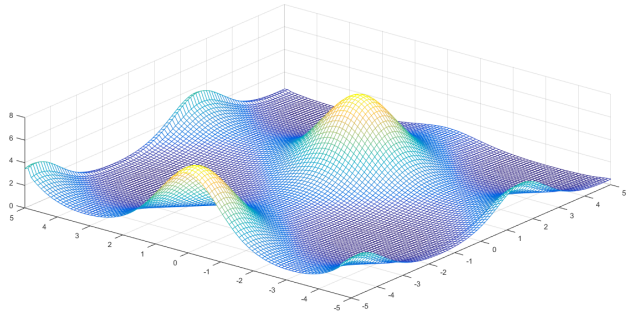


Figura 5.15: $e^{\sin(x)+\cos(y)}$

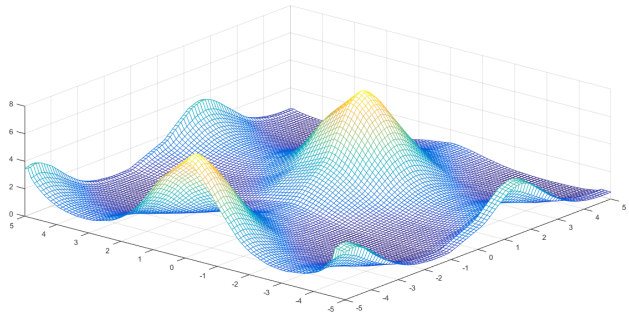
a) Spline Natural



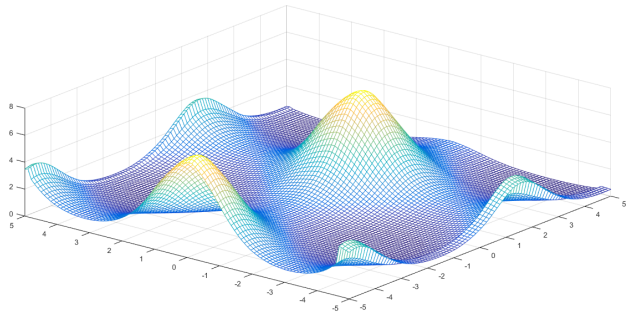
b) Spline Not-A-Knot



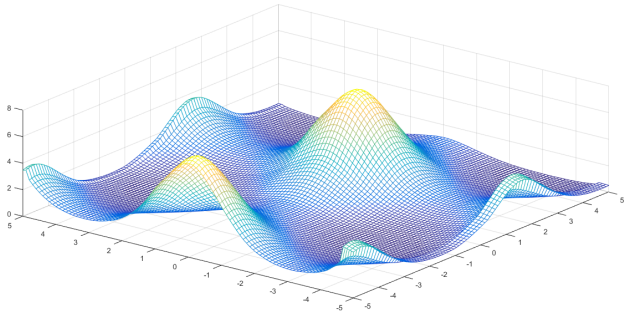
c) Lineal



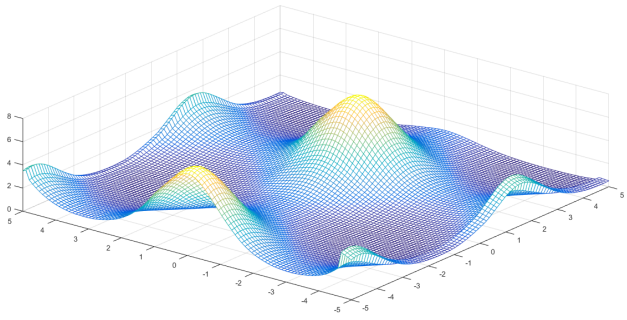
d) Akima



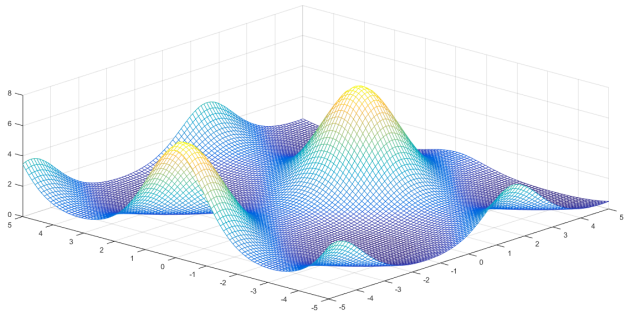
e) Restringido



f) Monótono



g) Bivariado de Akima



4. $f(x,y) = \sin(\sqrt{x^2+y^2})$

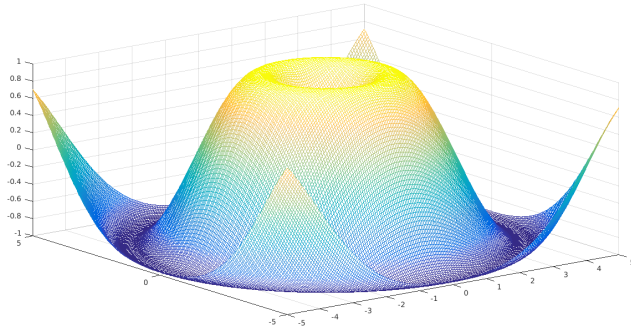
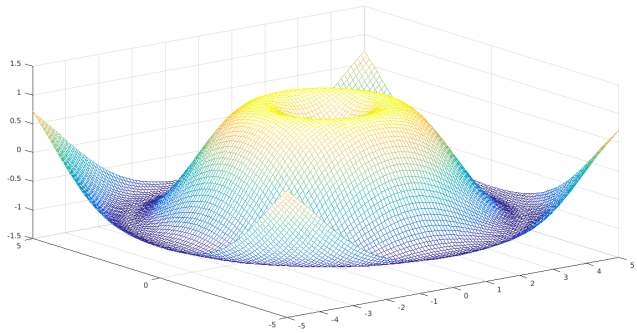
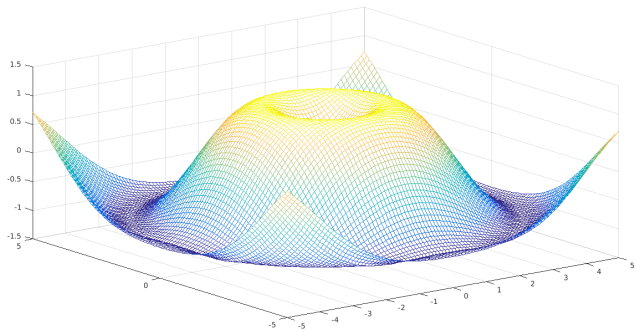


Figura 5.16: $\sin(\sqrt{x^2+y^2})$

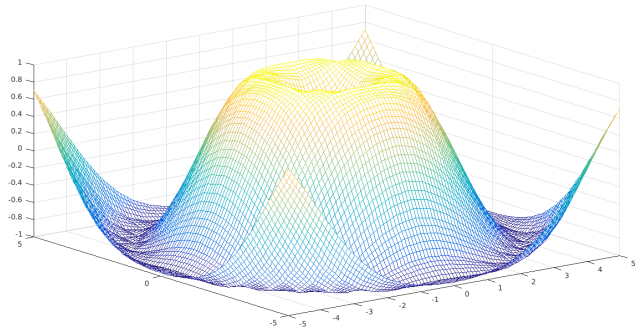
a) Spline Natural



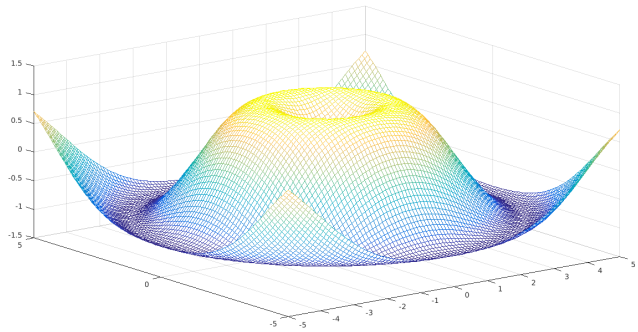
b) Spline Not-A-Knot



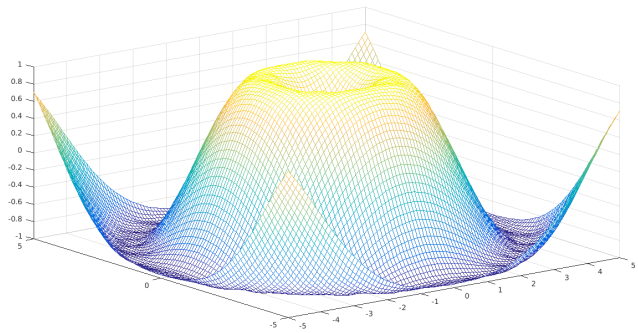
c) Lineal



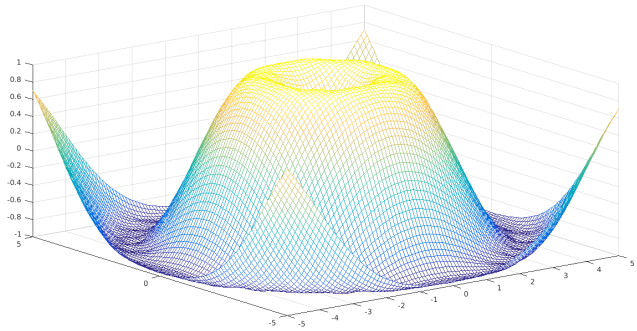
d) Akima



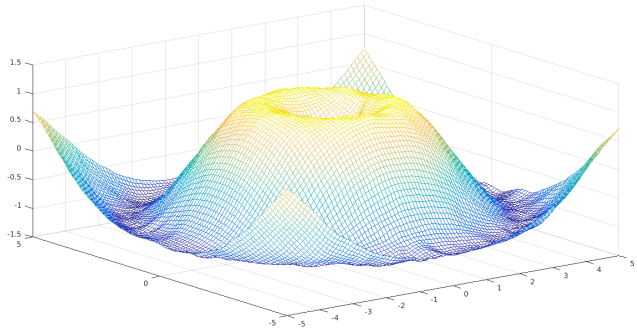
e) Restringido



f) Monótono



g) Bivariado de Akima



5.2.3. Interpolantes Óptimos

Una vez analizados los datos y las gráficas, podemos buscar interpolantes óptimos para las funciones analíticas.

En las gráficas con secciones no "suaves" observamos que los interpolantes IBMIU Spline y Bivaridado de Akima, comienzan a realizar curvas pronunciadas que se alejan de la superficie original, sin embargo los interpolantes IBMIU Hermite Cúbico, reducen el error en las secciones no "suaves".

Mediante las gráficas y los datos analizados, es posible realizar algunas observaciones, podemos notar, que para el caso de la IBMIU, los interpolantes Spline reducen mejor el error que los de tipo Hermite Cúbico en funciones suaves, y a que el Spline contempla la segunda derivada para su construcción, mientras que los interpolantes de tipo Hermite Cúbico sólo la primera derivada, veamos en la siguiente gráfica el comportamiento de los dos tipos de interpolante

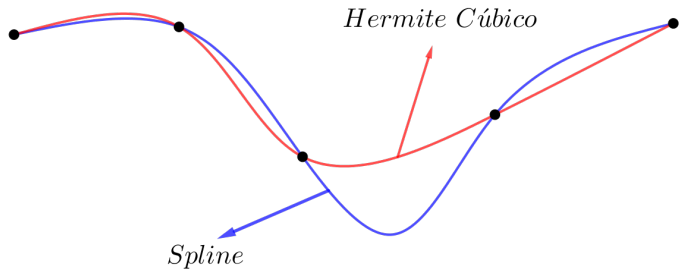


Figura 5.17: Spline y Hermite Cúbico

como se ve en la gráfica, el interpolante Spline suele tener curvas más pronunciadas en ciertas secciones, mientras que el Hermite Cúbico, no presenta esta característica, si lo calculamos en IBMIU, es posible notarlo en la gráficas y los datos en histogramas y tablas, resultado de las características propias de los interpolantes.
El caso del interpolante Bivariado de Akima, vemos que se reduce aún más el error, gracias a la construcción del método, que aproxima hasta la segunda derivada parcial $(z_x, z_y, z_{xx}, z_{xy}, z_{yy})$, para cada punto del triángulo al que pertenece un polinomio de grado cinco.

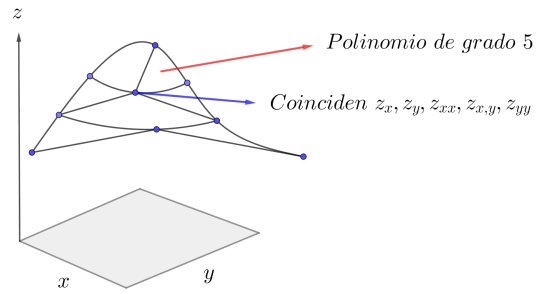


Figura 5.18: Spline y Hermite Cúbico

Otro aspecto importante además del error para evaluar la eficiencia de un método, es el costo numérico de cada interpolante, por un lado, la IBMIU tiene un costo menor, ya que consiste en aplicar un mismo método de interpolación univariado varias veces para aproximar un valor bivariado, haciendo con más rapidez el cálculo para varios puntos, por otro lado el método Bivariado de Akima reduce mejor el error, pero el costo es más alto, debido a que su construcción teórica requiere de varios procesos (triangulación, aproximación a las derivadas parciales, etc.).

En conclusión, podemos decir que para las funciones analíticas el interpolante **Bivariado de Akima** es el mejor, ya que reduce el error mejor que en otros interpolantes, sin embargo, si la función en la mayor de su estructura es "no suave" entonces el mejor estimador es el interpolante **IBMIU Akima** ya que al observar las gráficas y los datos vemos reduce mejor el error que los otros interpolantes de tipo Hemite Cúbico.

5.3. Fluidos en Yacimientos de Petróleos

Los datos utilizados para su análisis fueron proporcionados por la empresa Grupo SSC S.A. de C.V. en donde se desarrolló un modulo computacional PVT PCTSAG (Programa de Caracterización Termodinámica de Sistemas Aceite-Gas). En los modulos PVT se calculan propiedades termodinámicas, equilibrio de fases y algunas propiedades mecánicas. Los datos de las tablas a analizar provienen de solucionar para un conjunto de puntos iniciales la ecuación de estado Peng-Robinson [8]:

$$P = \frac{RT}{(v-b)} - \frac{a(t)}{(v^2+2vb-b^2)}$$

en el modulo PCTSAG. Los fluidos de yacimientos de petróleo y gas son mezclas multicomponentes constituidas principalmente de hidrocarburos. Los hidrocarburos presentes van del metano hasta hidrocarburos de alto peso molecular.

A continuación se describen de manera muy general, las propiedades de los fluidos que se analizarán con los interpolantes, propiedades dependen de una presión (PT: Fuerza que ejerce una gas, líquido o sólido sobre una superficie) y una temperatura (TM: Magnitud escalar con al que se mide el nivel térmico) determinada.

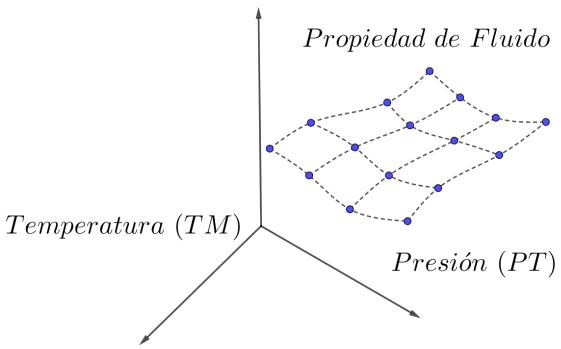


Figura 5.19

1. Densidad de Gas (ROG): Cantidad de moléculas gas que ocupan en un volumen determinado.

2. Densidad de Líquido (ROHL): Relación que existe entre el volumen y la masa del líquido.

3. Fracción Masa de Gas (RS): Fracción de masa del gas con respecto a la masa total de la mezcla.

4. Viscosidad de Gas (VISG): Resistencia a fluir de un gas.

5. Viscosidad de Líquido (VISHL): Resistencia a fluir de un líquido.

6. Capacidad Calorífica de Gas (CPG): Cantidad de energía necesaria para aumentar la temperatura de un gas.

7. Capacidad Calorífica de Líquido (CPHL): Cantidad de energía necesaria para aumentar la temperatura de un líquido.

8. Entalpía de gas (HG): Cantidad de energía contenida en un gas.

9. Entalpía de líquido (HHG): Cantidad de energía contenida en un líquido .

10. Conductividad Térmica de Gas (TCG): Capacidad de conducción de calor de una gas.

11. Conductividad Térmica de Líquido (TCHL): Capacidad de conducción de calor de un líquido.

12. Tensión interfacial del gas-líquido (SIGGHL): Energía que es el resultado de la diferencia del grado de atracción de las moléculas de la superficie del líquido con la del grado de atracción que existe en las moléculas del gas.

13. Entropía de Gas (SEG): Nivel de orden del sistema a nivel molecular del gas.

14. Entropía de Líquido (SEG): Nivel de orden del sistema a nivel molecular del líquido.

5.3.1. Análisis de Datos

Hasta el momento, se han estudiado métodos de interpolación y analizado como se comportan al ingresar puntos en funciones analíticas, ahora se buscará el interpolante óptimo, que aproxime los datos de las distintas superficies generadas propiedades de los fluidos.

Para realizar los cálculos, tomando en cuenta los requerimientos teóricos en los fluidos, se buscan dos propiedades importantes que debe cumplir el interpolante

1. Tiempo requerido para calcular un punto (por cuestiones de optimización).

2. El interpolante no debe hacer grandes cambios en la estructura de la superficie original.

3. El interpolante debe suavizar las superficies sin hacer grandes curvas en los "cambios de fase".

Tiempo de ejecución

A continuación se muestran los tiempos de ejecución de cada método bivariado (IBMIU y Bivariado de Akima), calculando un puntos interpolando dos mil quinientos datos (número de datos base de las propiedades de los fluidos).

Cuadro 5.11: Tiempo de ejecución de cada método

Interpolante	**Tiempo en segundos**
Spline Natural (IBMIU)	0.00205349960893
Spline Not-A-Knot (IBMIU)	0.00190553816272
Lineal (IBMIU)	0.00092454522168
Akima (IBMIU)	0.00180418884841
Restringido (IBMIU)	0.00152152261736
Monótono (IBMIU)	0.00234086981659
Bivariado de Akima	2.14729700000000

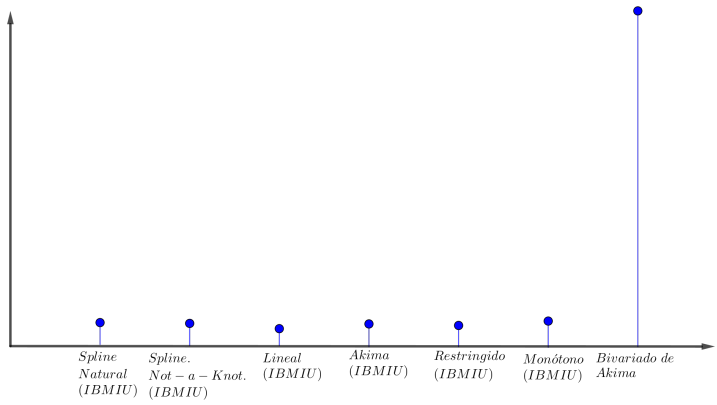


Figura 5.20: Gráfica de tiempos en segundos de los interpolantes

Podemos notar que el interpolante bivariado de Akima es el que consume más tiempo de ejecución, es importante mencionar que dicho tiempo fue calculado tomando en cuenta sólo tres puntos cercanos para estimar las derivadas parciales, si se requieren más puntos cercanos para la estimación, el tiempo de ejecución crece.

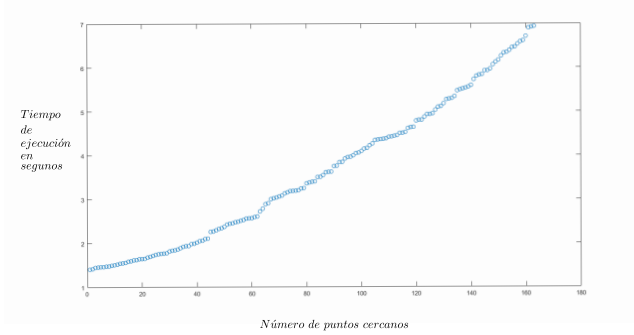


Figura 5.21: Gráfica del tiempo de ejecución en relación a puntos cercanos

Por lo anterior, tomando en cuenta los tiempos de ejecución, el interpolante Bivariado de Akima no podría considerarse, sin embargo es necesario analizar su comportamiento en las superficies de las propiedades de los fluidos para verificar si los resultados obtenidos gráficamente podrían hacer que el tiempo de ejecución no sea una determinante para su elección.

Análisis Gráfico de los Interpolantes

Anteriormente, se había mencionado, que el interpolante no debe hacer grandes cambios en la estructura de las superficies originales en las propiedades

de los fluidos, las estructuras pueden ser clasificadas en tres grupos

1. Superficies regulares

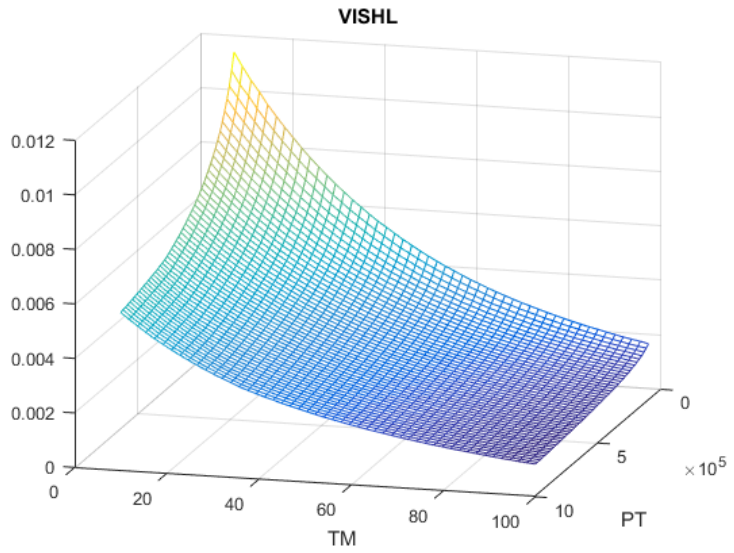


Figura 5.22: Propiedad VISHL

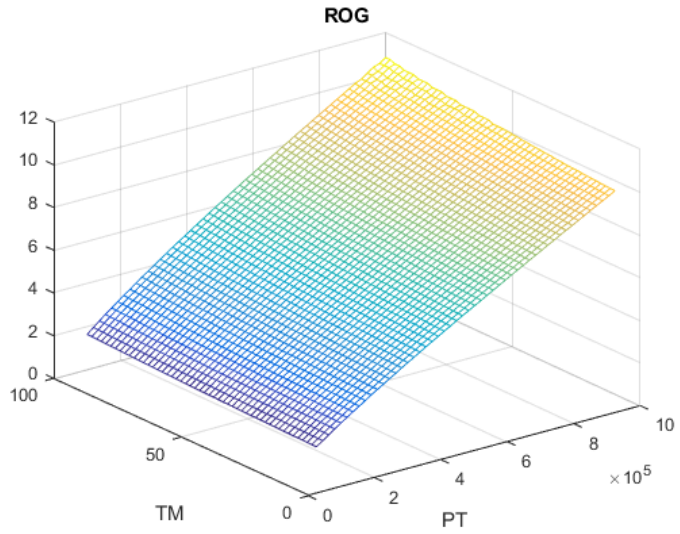


Figura 5.23: Propiedad ROG

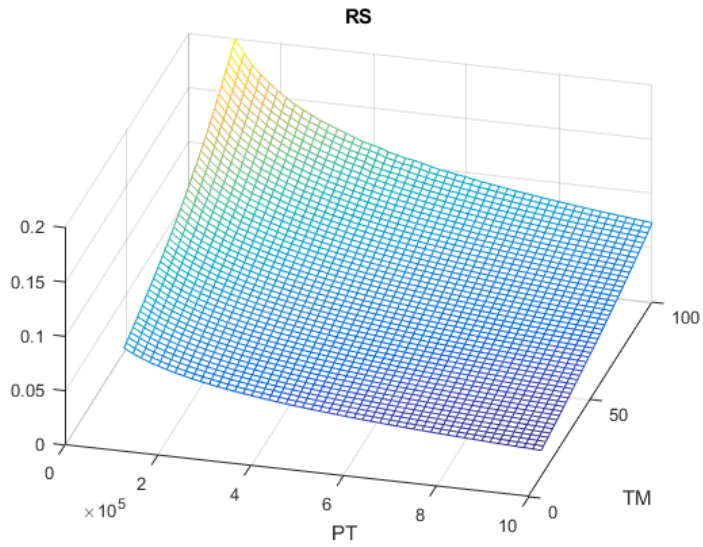


Figura 5.24: Propiedad RS

2. Irregulares

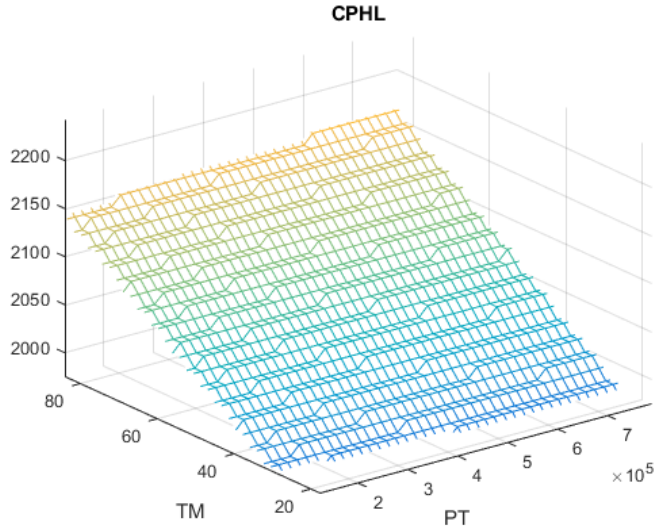


Figura 5.25: Propiedad CPHL

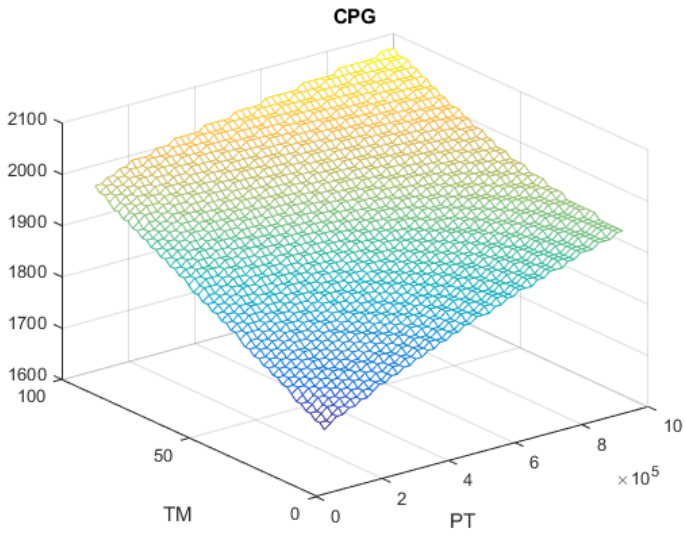


Figura 5.26: Propiedad CPG

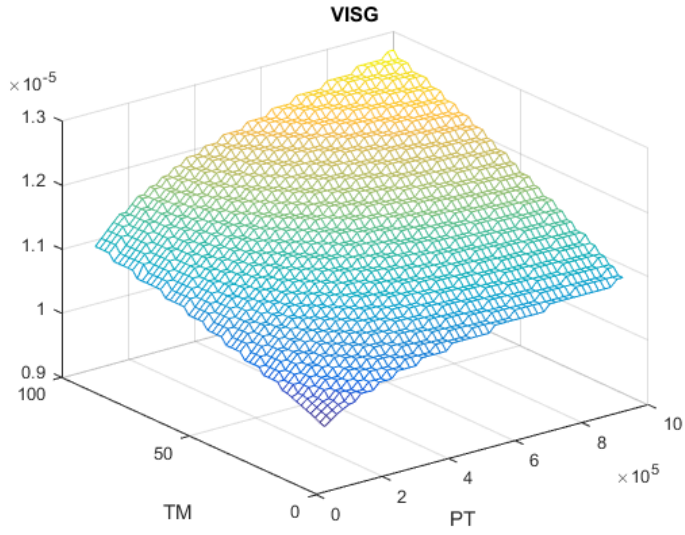


Figura 5.27: Propiedad VISG

3. Con cambios de fase: Esta característica, la analizaremos de manera gráfica, observemos como es el comportamiento de las superficies generadas por las propiedades

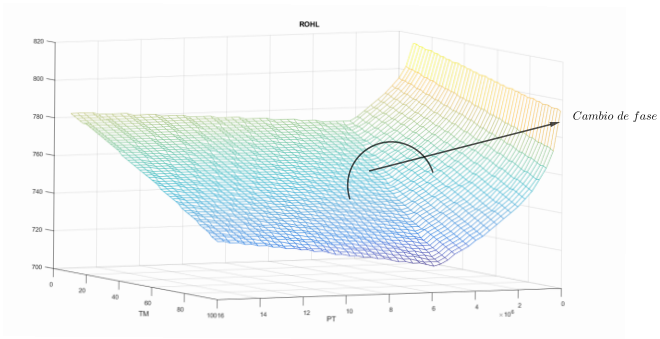


Figura 5.28: Propiedad ROHL

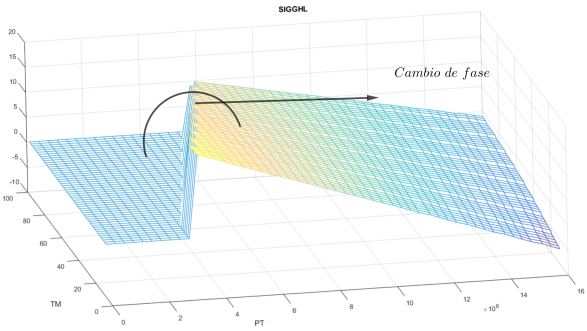


Figura 5.29: Propiedad SIGGHL

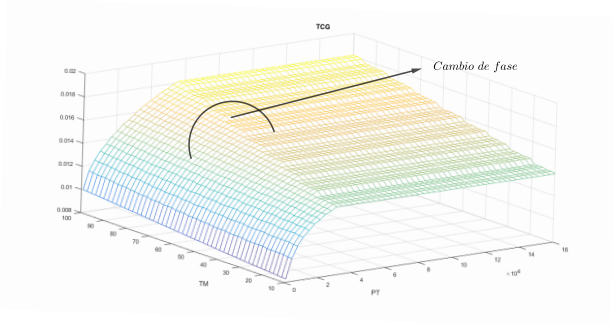


Figura 5.30: Propiedad TCG

es importantes destacar que en las superficies puede cambiar de regulares a irregulares después de un cambio de fase.

El método actualmente utilizado al interpolar dichas superficies es el interpolante lineal, ahora analizaremos el comportamiento de los interpolantes en las tres características antes mencionadas divididos en tres grupos, IBMIU Spline Cúbico, IBMIU Hermite Cúbico y Bivariado de Akima, primero observaremos su comportamiento gráfico sobre superficies regulares, realizando la interpolación sobre la propiedad VISHL

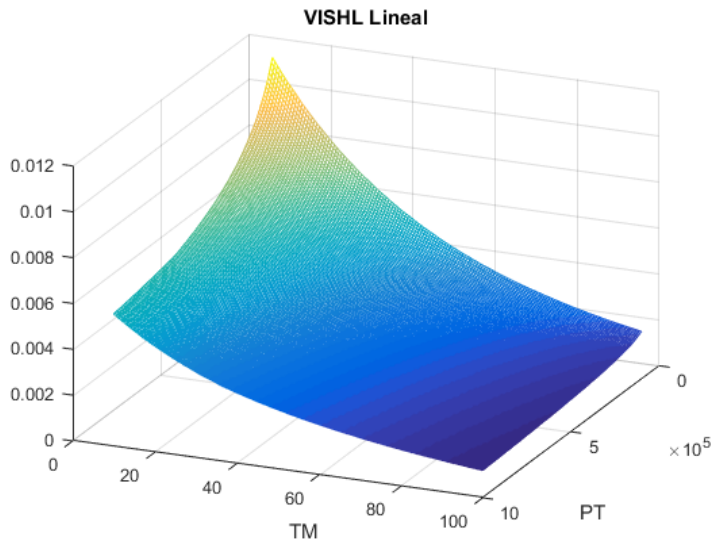


Figura 5.31: IBMIU Lineal CPHL

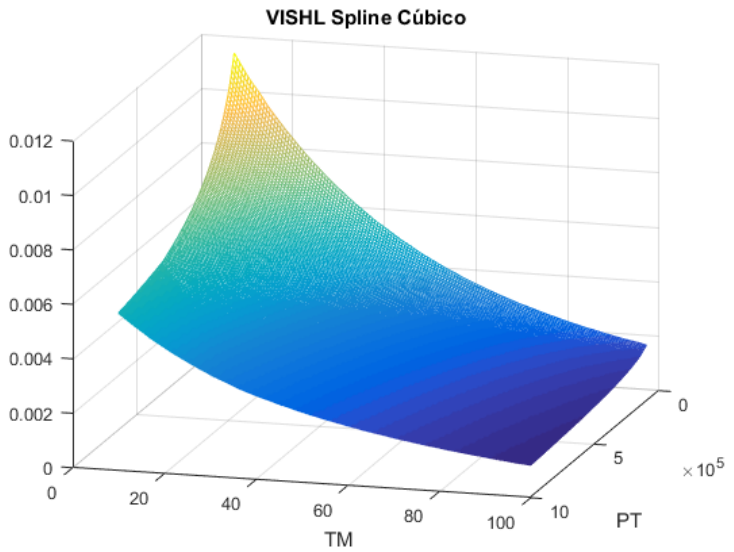


Figura 5.32: IBMIU Spline Cúbico CPHL

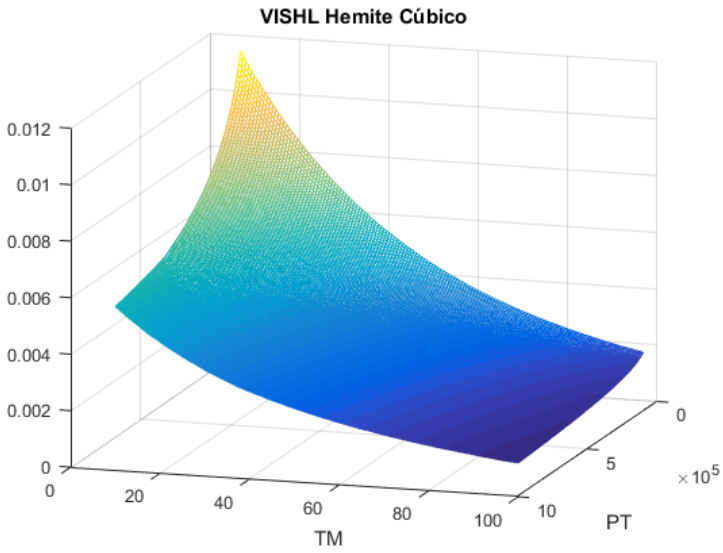


Figura 5.33: IBMIU Hermite Cúbico CPHL

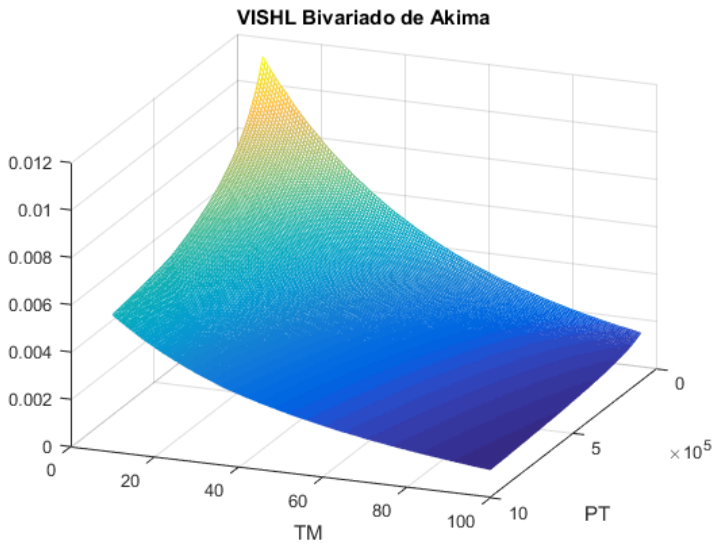


Figura 5.34: Bivariado de Akima CPHL

Es importante notar que no existe un cambio significativo en la suavización de los interpolantes sobre las superfices regulares, siendo los interpolates Lineal, IBMIU Spline Cúbico, IBMIU Hermite Cúbico y BIvariado de Akima métodos candidatos para este tipo de superficies, ahora analicemos su comportamiento sobre las superficies irregulares, tomaremos las gráfica generadas con la propiedad CPHL.

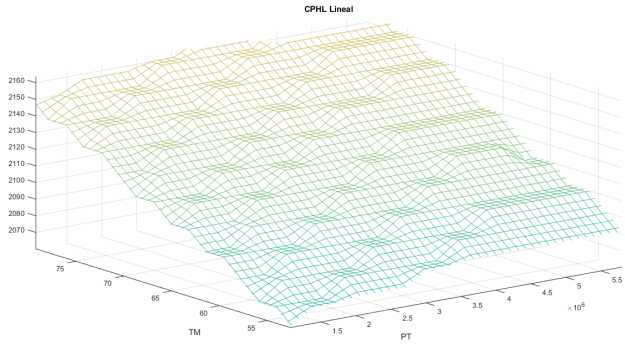


Figura 5.35: IBMIU Lineal CPHL

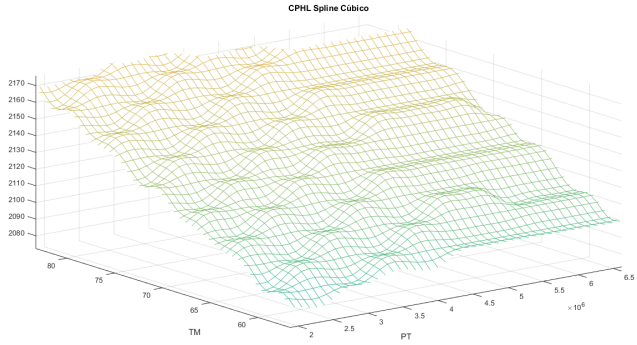


Figura 5.36: IBMIU Spline Cúbico CPHL

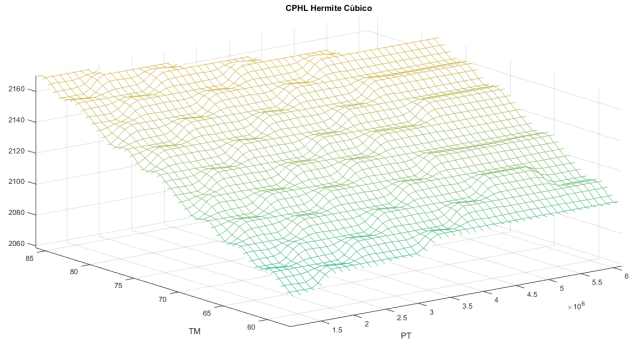


Figura 5.37: IBMIU Hermite Cúbico CPHL

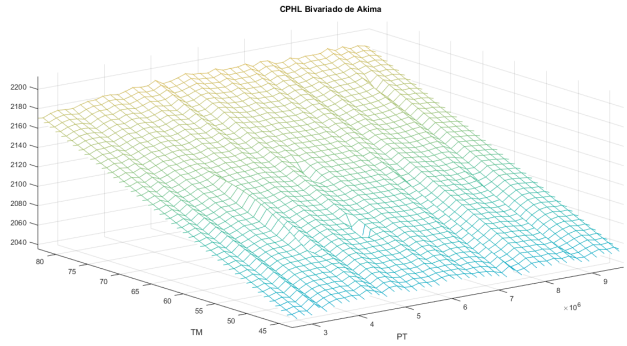


Figura 5.38: Bivariado de Akima CPHL

Al observar las distintas gráficas, notamos que en el caso de las IBMIU Hermite Cúbico e IBMIU Spline Cúbico hubo un mejoramiento en la suavización en las superficies al comparar con la interpolación lineal.

En el caso de la Interpolación bivariada de Akima, el comportamiento del interpolante no ayudó a mejorar la suavización, ya que generó una superficie con algunos "picos" y curvas muy pronunciadas como se muestra en la figura.

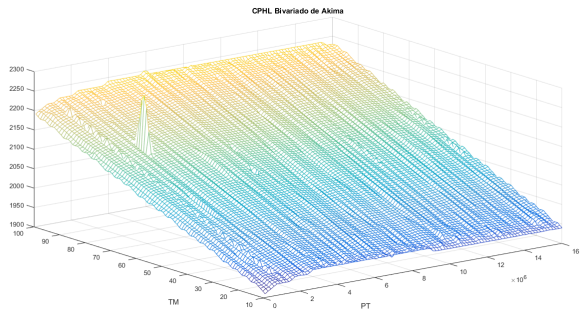


Figura 5.39: Bivariado de Akima CPHL

Por la observación anterior, no podemos contemplar al interpolante bivariado de Akima, ya que no cumple con uno de los requerimientos básicos de no generar grandes cambios en la estructura de la superficie original. Ahora debemos ver el comportamiento de los interpolantes en los "cambios de fase", veamos las gráficas de los interpolantes Spline Cúbico y Hermite Cúbico para analizar su comportamiento sobre la propiedad SIGGHL.

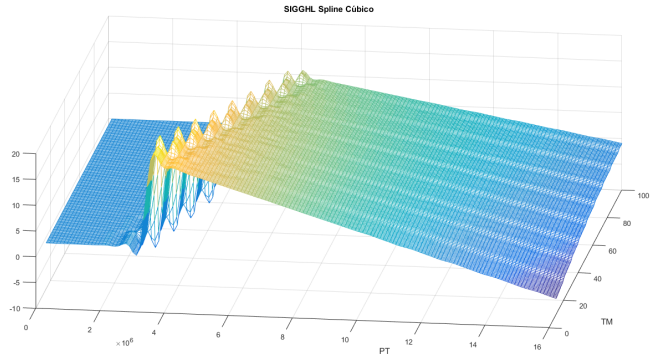


Figura 5.40: IBMIU Spline Cúbico “cambio de fase”

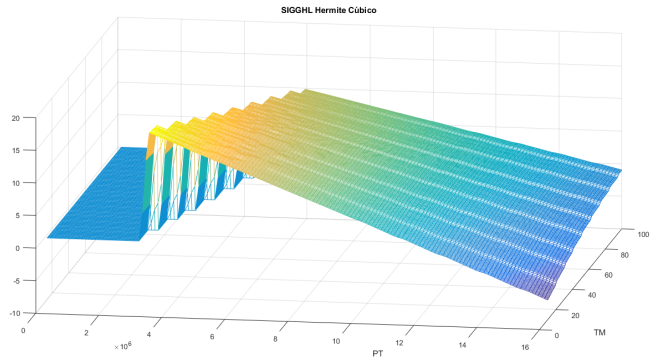


Figura 5.41: IBMIU Hermite Cúbico “cambio de fase”

Como podemos observar el interpolante Spline realiza grandes curvas en los “cambios de fase” mientras que en interpolante Hermite Cúbico suaviza las superficies sin realizar curvas muy pronunciadas en los “cambios de fase”.

Seleccionando el interpolante

Ahora observemos que los mejores métodos tomando en cuenta las condiciones gráficas son los que corresponden al IBMIU Hermite Cúbico (Akima, Restringido y Monónoto) ya que logran suavizar las superficies sin hacer grandes curvas en los “cambios de fase” además de no cambiar de manera significativa en la estructura original de las superficies, ahora analizaremos el criterio del tiempo de ejecución para los interpolantes

Cuadro 5.12: Tiempo de ejecución de cada método

Interpolante	**Tiempo en segundos**
Akima (IBMIU)	0.00180418884841
Restringido (IBMIU)	0.00152152261736
Monótono (IBMIU)	0.00234086981659

Observemos que el mejor tiempo de ejecución lo tiene el interpolante IBMIU Restringido, sin embargo el método tiene el problema de no estar definido si alguna pendientes $s_i = 0$, lo cual puede provocar errores al momento de hacer los cálculos ya que en la figura 5.40 vemos una sección de la superficie donde parte de su estructura parece un plano con pendientes $s_i = 0$.

Debido a la observación mencionada, el siguiente mejor tiempo es el Método IBMIU de Akima, qué es un método bivariado óptimo en cuestión de suavización sobre superficies regulares e irregulares y tiempo requerido, además de estar perfectamente definido para las cualquier pendiente, por tal motivo concluimos que este método IBMIU **Bivarido de Akima** es el mejor para poder interpolar las superficies en las propiedades de los fluidos, mejorando la suavización de la interpolación lineal y sin desbordarse en los "cambios de fase", conservando la estructura de las superficies originales, además de la eficiencia en el tiempo de la ejecución.

Apéndice A

Spline Cúbico,una forma alternativa

Es posible dar una contrucción alterna a la vista previamente en el Capitulo 2. Teniendo $(x_0, y_0), (x_1, y_1), ..., (x_n, y_n)$, $n+1$ puntos en $\mathbb{R}^2$ y cumplen que $x_0 < x_1 < \cdots < x_j < \cdots < x_{n-1} < x_n$,llamamos S_j con $j = 0, 1, ..., n-1$ a los polinomios de grado 3 que pasan por cada punto (x_i, y_i) para $i = 0, 1, ..., n$ y que además cumplen:

1. $S_{i-1}(x_i) = y_i = S_i(x_i)$
2. $S'_{i-1}(x_i) = S'_i(x_i)$
3. $S''_{i-1}(x_i) = S''_i(x_i)$

con $i = 1, 2, ..., n-1$.

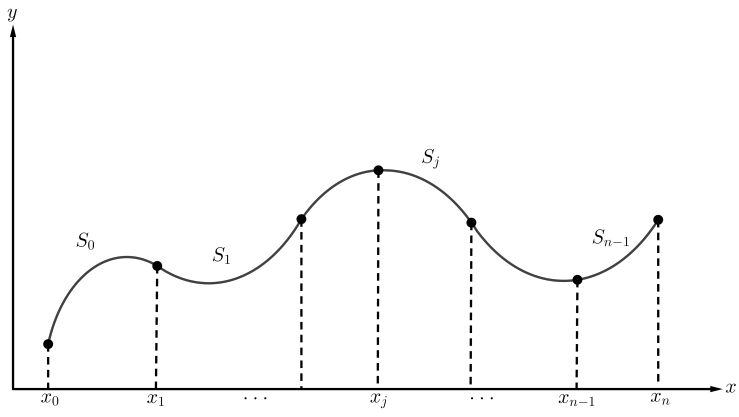


Figura A.1

Usando las condiciones 1,2 y 3 construiremos una expresión para $S_i(x)$, la condición 3, nos permite dar una condición de continuidad que llamaremos z_i

$$\lim_{x \to x_i^-} S''(x) = z_i = \lim_{x \to x_i^+} S''(x)$$

para $i = 1, 2, ..., n-1$. Si $S_i(x)$ es un polinomio de grado 3, entonces $S_i''(x)$ es una función lineal que podemos definir a partir de z_i usando la ecuación de la recta

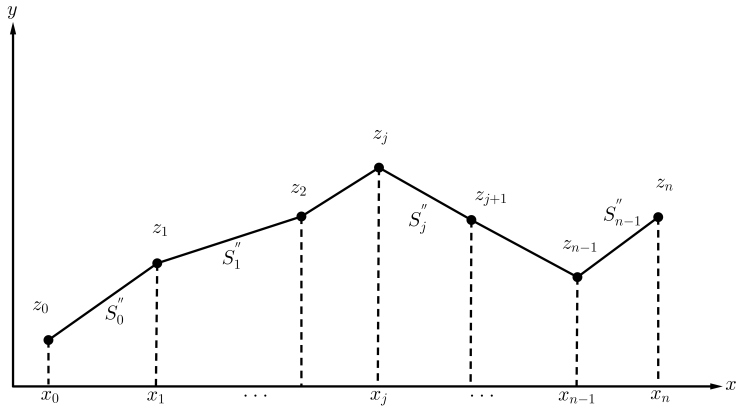


Figura A.2

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

donde $h_i = x_{i+1} - x_i$. Al integrar la expresión obtenemos

$$\begin{aligned} S_i'(x) &= \int \frac{z_{i+1}}{h_i}(x - x_i) + \frac{z_i}{h_i}(x_{i+1} - x)\, dx \\ &= \frac{z_{i+1}}{2h_i}(x - x_i)^2 - \frac{z_i}{2h_i}(x_{i+1} - x)^2 + C1 \end{aligned}$$

integrando nuevamente

$$\begin{aligned} S_i(x) &= \int \frac{z_{i+1}}{2h_i}(x - x_i)^2 - \frac{z_i}{2h_i}(x_{i+1} - x)^2 + C1\, dx \\ &= \frac{z_{i+1}}{6h_i}(x - x_i)^3 + \frac{z_i}{6h_i}(x_{i+1} - x)^3 + Cx + D \end{aligned}$$

podemos modificar las constantes de $S_i(x)$, ya que esto no afectará a $S_i'(x)$, entonces haciendo

$$\begin{aligned} C &= (C_1 - C_2) \text{ y} \\ D &= C_2 x_{i+1} - C_1 x_i \end{aligned}$$

obtenemos

$$S_i(x) = \frac{z_{i+1}}{6h_i}(x - x_i)^3 + \frac{z_i}{6h_i}(x_{i+1} - x)^3 + C_1(x - x_i) + C_2(x_{i+1} - x)$$

usando la condición 1 obtenemos que $S_i(x_i) = y_i$ y $S_i(x_{i+1}) = y_{i+1}$, con lo cual

$$y_i = \frac{z_i}{6h_i} + C_2 h_i$$

despejando C_2

$$C_2 = \frac{y_i}{h_i} - \frac{z_i}{6}h_i$$

de la misma forma

$$y_{i+1} = \frac{z_{i+1}}{6h_i} + C_1 h_i$$

despejando C_1

$$C_1 = \frac{y_{i+1}}{h_i} - \frac{z_{i+1}}{6}h_i$$

y la expresión final de $S_i(x)$ es

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

derivando la nueva expresión $S_i(x)$

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

la condición 2 nos da como resultado que $S_{i-1}^{'}(x_i) = S_i^{'}(x_i)$ por lo tanto se obteniene la siguiente igualdad

$$-\frac{h_i}{3}z_i - \frac{h_i}{6}z_{i+1} - \frac{y_i}{h_i} + \frac{y_{i+1}}{h_i} = \frac{h_{i-1}}{3}z_i + \frac{h_{i-1}}{6}z_{i-1} - \frac{y_{i-1}}{h_{i-1}} + -\frac{y_i}{h_{i-1}}$$

reagrupando obtenemos

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

para $i = 1, 2, ..., n-1$, con lo cual obtenemos el sistema de ecuaciones $Ax = b$

$$A = \begin{pmatrix} h_0 & 2(h_0+h_1) & h_1 & \ddots & & & \vdots \\ 0 & h_1 & 2(h_1+h_2) & h_2 & \ddots & & 0 \\ 0 & 0 & h_2 & 2(h_2+h_3) & h_3 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & & \\ & & & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} & \end{pmatrix}$$

$$x = \begin{pmatrix} z_0 \\ z_1 \\ z_2 \\ \\ \vdots \\ \\ z_n \end{pmatrix}$$

$$b = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ \\ v_{n-1} \end{pmatrix}$$

donde

$$b_i = \frac{6}{h_i}(y_{i+1} - y_i)$$
$$v_i = b_i - b_{i-1}$$

para $i = 1, 2, ..., n-1$, el sistema obtenido, no es cuadrado, para obtener un sistema cuadrado es necesario contar con condiciones de frontera para completar el sistema, dependiendo de estas condiciones, el Spline Cúbico recibirá diferentes nombres.

A.1. Spline Cúbico Natural

En este caso la condición de frontera es

$$S_0''(x_0) = S_n'' - 1(x_n) = 0$$

con lo cual

$$\frac{z_0}{h_0}h_0 = 0$$
$$\frac{z_n}{h_{n-1}}h_{n-1} = 0$$

y podemos completar el sistema $Ax = b$ para que sea cuadrado, resolviendo el sistema para obtener los valores de los coeficientes de cada polinomio $S_i(x)$, en este caso

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & \cdots & 0 \\ h_0 & 2(h_0+h_1) & h_1 & \ddots & & & \vdots \\ 0 & h_1 & 2(h_1+h_2) & h_2 & \ddots & & 0 \\ 0 & 0 & h_2 & 2(h_2+h_3) & h_3 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & & \\ & & & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} & 0 \\ 0 & \cdots & \cdots & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$x = \begin{pmatrix} z_0 \\ z_1 \\ z_2 \\ \\ \vdots \\ \\ z_n \end{pmatrix}$$

$$b = \begin{pmatrix} 0 \\ v_1 \\ v_2 \\ \vdots \\ \\ v_{n-1} \\ 0 \end{pmatrix}$$

A.2. Spline Cúbico Completo

Las condiciones de frontera para este Spline son las siguientes:

$$S_0'(x_0) = f'(x_0)$$
$$S_{n-1}'(x_n) = f'(x_n)$$

con lo cual obtenemos el siguiente resultado

$$f'(x_0) = S_0'(x_0) = -\frac{z_0}{2h_0}h_0^2 + \frac{y_1}{h_0} - \frac{z_1}{6}h_0 - \frac{y_0}{h_0} + \frac{z_0}{6}h_0$$

reordenando términos

$$2h_0z_0 + h_0z_1 = \tfrac{6}{h_0}(y_1 - y_0) - 6f'(x_0)$$

con el mismo rezonamiento

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

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

con lo anteriormente obtenido completamos el sistema

$$A = \begin{pmatrix} 2h_0 & h_0 & 0 & 0 & \cdots & \cdots & 0 \\ h_0 & 2(h_0+h_1) & h_1 & \ddots & & & \vdots \\ 0 & h_1 & 2(h_1+h_2) & h_2 & \ddots & & 0 \\ 0 & 0 & h_2 & 2(h_2+h_3) & h_3 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & & \\ & & & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} & 0 \\ 0 & \cdots & \cdots & 0 & 0 & h_{n-1} & 2h_{n-1} \end{pmatrix}$$

$$x = \begin{pmatrix} z_0 \\ z_1 \\ z_2 \\ \\ \vdots \\ \\ z_n \end{pmatrix}$$

$$b = \begin{pmatrix} \frac{6}{h_0}(y_1 - y_0) - 6f'(x_0) \\ v_1 \\ v_2 \\ \vdots \\ \\ v_{n-1} \\ 6f'(x_n) - \frac{6}{h_{n-1}}(y_n - y_{n-1}) \end{pmatrix}$$

A.3. Spline Cúbico Not-a-Knot

Las condiciones de frontera son

$$S_0'''(x_0) = S_1'''(x_1)$$
$$S_{n-2}'''(x_{n-1}) = S_{n-1}'''(x_n)$$

Calculando $S_i'''(x)$ obtenemos

$$S_i'''(x) = \frac{z_{i+1}}{h_i} - \frac{z_i}{h_i}$$

usando las condiciones de frontera

$$\frac{z_1}{h_0} - \frac{z_0}{h_0} = \frac{z_2}{h_1} - \frac{z_1}{h_1}$$
$$-h_1 z_0 + (h_0 + h_1)z_1 - h_0 z_2 = 0$$

de manera similar

$$\frac{z_{n-1}}{h_{n-2}} - \frac{z_{n-2}}{h_{n-2}} = \frac{z_n}{h_{n-1}} - \frac{z_{n-1}}{h_{n-1}}$$
$$-h_{n-1} z_{n-2} + (h_{n-2} + h_{n-1})z_{n-1} - h_{n-2} z_2 = 0$$

por lo tanto el sistema se modifica de la siguiente manera

$$A = \begin{pmatrix} -h_1 & h_0 + h_1 & -h_0 & 0 & \cdots & \cdots & 0 \\ h_0 & 2(h_0 + h_1) & h_1 & \ddots & & & \vdots \\ 0 & h_1 & 2(h_1 + h_2) & h_2 & \ddots & & 0 \\ 0 & 0 & h_2 & 2(h_2 + h_3) & h_3 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & & \\ & & & h_{n-2} & 2(h_{n-2} + h_{n-1}) & h_{n-1} & 0 \\ 0 & \cdots & \cdots & 0 & -h_{n-1} & h_{n-1} + h_{n-2} & -h_{n-2} \end{pmatrix}$$

$$x = \begin{pmatrix} z_0 \\ z_1 \\ z_2 \\ \\ \vdots \\ \\ z_n \end{pmatrix}$$

$$b = \begin{pmatrix} 0 \\ v_1 \\ v_2 \\ \vdots \\ \\ v_{n-1} \\ 0 \end{pmatrix}$$

Apéndice B

Códigos

A continuación se presentan los códigos de programación de cada uno de los interpolantes mencionados en la tesis, dichos códigos fueron realizados en el softwere numérico Matlab en su versión 2016a.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                          %
%Interpolación de Lagrange %
%                          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Parámetros de Entrada:

%x,y:Vectores que guardan los números de los pares ordenado (xi,yi)
%    para i=0,1,2,....,n donde x0<x1<x2<...<xn.

%newx:Vector de elementos para los cuales se estimarán mediante la
Interpolación
%     Lagrange.

function[newy]=ILagrange(x,y,newx)
    n1=length(x);%Obtenemos el número de elementos de x.
    n2=length(newx);%Obtenemos el número de elementos de newx.
    Lk=zeros(n1,n2);%Matriz que guarda los valores Lk.
    prodLk=1;%Variable para realizar el producto Lk.
    for i=(1:n2)
        for k=(1:n1)
            for j=(1:n1)
                if(j~=k)
                    prodLk=prodLk*((newx(i)-x(j))/(x(k)-x(j)));%Obtenemos cada
                    %valor Lk.
                end
            end
            Lk(k,i)=prodLk;%%Guardamos el valor Lk correspondiente.
            prodLk=1;
        end
    end
    %Obtenemos los nuevos valores de newy.
    newy=(y'*Lk)';
end

%Parámetros de Salida:

%newy:Nuevos puntos estimados mediante el polinomio de Lagrange.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                          %
%Interpolación de Newton   %
%                          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Parámetros de Entrada:

%x,y:Vectores que guardan los números de los pares ordenado (xi,yi)
%    para i=0,1,2,....,n donde x0<x1<x2<...<xn.

%newx:Vector de elementos para los cuales se estimarán mediante la
Interpolación
%     Newton.

function[newy]=INewton(x,y,newx)
    n1=length(x);%Obtenemos el número de elemntos de x.
    n2=length(newx);%Obtenemos el número de elemntos de newx.
    PNewton=zeros(n1-1,n2);%Matriz que guardará los valores de polinomio de
    Newton.
    producto=1;%Variable para realizar el producto del polinomio.
    %Construimos el polinomio de Newton para cada newx.
    for i=(1:n2)
        for k=(1:n1-1)
            producto=producto*(newx(i)-x(k));%Obtenemos el producto.
            PNewton(k,i)=producto;%Guadamos los valores del polinomio.
        end
        producto=1;
    end
    %Obtenemos los ak (Diferencias divididas).
     ak=DDivididasN(x,y);
    %Reajustamos la matriz PNewton.
    PNewton=[ones(1,n2);PNewton];
    newy=(ak'*PNewton)';%Obtenemos los nuevos valores de y.
end

%Parámetros de Salida:

%newy:Nuevos puntos estimados mediante el polinomio de Lagrange.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                   %
%Diferencias Divididas de Newton    %
%                                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Parámetros de Entrada:

%x,y:Vectores que guardan los números de los pares ordenado (xi,yi)
%    para i=0,1,2,....,n donde x0<x1<x2<...<xn.

function[ak]=DDivididasN(x,y)
    n=length(x);%Obtenemos el número de elemntos de x.
    ak=zeros(n,n);%Matriz que guarda las dierencias divididas de Newton.
    %Obtenemos la diferencias divididas de Newton.
    ak(:,1)=y;
    for i=(2:n)
        for j=(2:i)
            ak(i,j)=(ak(i,j-1)-ak(i-1,j-1))/(x(i)-x(i-(j-1)));
        end
    end
    %Obtenemos la diagonal de ak que son los valoores de los coeficientes.
    %del interpolante de Newton.
```

```
63          ak=diag(ak);
64      end
65
66      %Parámetros de Salida:
67
68      %ak:Vector que contiene los coeficientes del Interpolante de Newton.
69
70
71
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                           %
%Interpolación de Hermite  %
%                           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Parámetros de Entrada:

%x,y:Vectores que guardan los números de los pares ordenado (xi,yi)
%    para i=0,1,2,....,n donde x0<x1<x2<...<xn.


%dy:Vector que guarda las derivadas en los puntos(xi,yi).

%newx:Vector de elementos para los cuales se estimarán mediante la
Interpolación
%     de Hermite.

function[newy]=IHermite(x,y,dy,newx)
    n1=length(x);%Obtenemos el número de elemntos de x.
    n2=length(newx);%Obtenemos el número de elemntos de newx.
    PHermite=zeros(2*n1-1,n2);%Matriz que guardará los valores de polinomio
    de Hermite.
    producto=1;%Variable para realizar el producto del polinomio.
    %Obtenemos los coeficientes de Polinomio de Hermite.
    Q=DDividididasH(x,y,dy);
    %Construimos el polinomio de Hermite para cada newx.
    for i=(1:n2)
        for k=(1:n1-1)
            producto=producto*(newx(i)-x(k));
            PHermite(2*k-1,i)=producto;
            producto=producto*(newx(i)-x(k));
            PHermite(2*k,i)=producto;
        end
        producto=producto*(newx(i)-x(n1));
        PHermite(2*n1-1,i)=producto;
        producto=1;
    end
    %Reajustamos la matriz PHermite.
    PHermite=[ones(1,n2);PHermite];
    newy=(Q'*PHermite)';%Obtenemos los nuevos valores de y.

end

%Parámetros de Salida:

%newy:Nuevos puntos estimados mediante el polinomio de Lagrange.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                   %
%Diferencias Divididas de Hermite  %
%                                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Parámetros de Entrada:

%x,y:Vectores que guardan los números de los pares ordenado (xi,yi)
%    para i=0,1,2,....,n donde x0<x1<x2<...<xn.

%dy:Vector que guarda las derivadas en los puntos(xi,yi).

function[Q]=DDividididasH(x,y,dy)
    n=length(x);%Obtenemos el número de elemntos de x.
    z=zeros(2*n,1);%Vector que guarda los nuevos valores zi para.
```

```
63      Q=zeros(2*n,2*n);%Matriz que guarda las dierencias divididas de Newton.
64      %Obtenemos la diferencias divididas de Hermite.
65      for i=(1:n)
66          z(2*i-1)=x(i);
67          z(2*i)=x(i);
68          Q(2*i-1,1)=y(i);
69          Q(2*i,1)=y(i);
70          Q(2*i,2)=dy(i);
71          if(i>1)
72              Q(2*i-1,2)=(Q(2*i-1,1)-Q(2*i-2,1))/(z(2*i-1)-z(2*i-2));
73          end
74      end
75      for i=(3:2*n)
76          for j=(3:i)
77              Q(i,j)=(Q(i,j-1)-Q(i-1,j-1))/(z(i)-z(i-(j-1)));
78          end
79      end
80      %Obtenemos la diagonal de Q que son los valoores de los coeficientes
81      %del interpolante de Hermite.
82      Q=diag(Q);
83  end
84
85  %Parámetros de Salida:
86
87  %ak:Vector que contiene los coeficientes del Interpolante de Hermite.
88
89
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%
%                        %
%Interpolación Lineal    %
%                        %
%%%%%%%%%%%%%%%%%%%%%%%%%%

%Parámetros de Entrada:

%x,y:Vectores que guardan los números de los pares ordenado (xi,yi)
%    para i=0,1,2,....,n donde x0<x1<x2<...<xn.

%newx:Vector de elementos para los cuales se estimarán mediante la
Interpolación
%     Lineal.

function[newy]=ILineal(x,y,newx)
    n1=length(x);%Obtenemos el número de elementos de x.
    n2=length(newx);%Obtenemos el número de elementos de newx.
    newy=zeros(n2,1);%Vector que guardará los nuevos valores de y.

    %Calculamos la pendiente de la recta.
    s=zeros(n1-1,1);
    h=zeros(n1-1,1);
    for i=(1:n1-1)
        h(i)=x(i+1)-x(i);
        s(i)=(y(i+1)-y(i))/h(i);
    end
    %Encontramos el nuevo punto para determinar a que recta pertenece.
    for i=(1:n2)
        distancia=abs(min(x)-max(x));
        for j=(1:n1)
            if(abs(newx(i)-x(j))<distancia)
                distancia=abs(newx(i)-x(j));
                indx=j;
            end
        end
        %Aproximamos los nuevos puntos y.
        if(newx(i)<x(indx))
            indx=indx-1;
            newy(i)=y(indx)+s(indx)*(newx(i)-x(indx));
        elseif(newx(i)>x(indx))
            newy(i)=y(indx)+s(indx)*(newx(i)-x(indx));
        elseif((newx(i)-x(indx))<10^-8)
            newy(i)=y(indx);
        end
    end
end
%Parámetros de Salida:

%newy:Nuevos puntos estimados mediante Interpolación Lineal.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                        %
%Interpolación Spline Cúbico Natural     %
%                                        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Parámetros de Entrada:

%x,y:Vectores que guardan los números de los pares ordenado (xi,yi)
%    para i=0,1,2,....,n donde x0<x1<x2<...<xn.

%newx:Vector de elementos para los cuales se estimarán mediante la
Interpolación
%     Lagrange

function[newy]=SCNatural(x,y,newx)
    n1=length(x);%Obtenemos el número de elementos de x.
    n2=length(newx);%Obtenemos el número de elementos de newx.
    newy=zeros(n2,1);%Vector que guardará los nuevos valores de y.

    %Calcuamos los coeficientes de los polinomio.
    z=zeros(n1,1);
    h=zeros(n1-1,1);
    b=zeros(n1-1,1);
    u=zeros(n1-2,1);
    v=zeros(n1-2,1);
    for i=(1:n1-1)
        h(i)=x(i+1)-x(i);
        b(i)=6*(y(i+1)-y(i))/h(i);
    end
    u(1)=2*(h(1)+h(2));
    v(1)=b(2)-b(1);
    for i=(2:n1-2)
        u(i)=2*(h(i)+h(i+1))-(h(i)^2)/u(i-1);
        v(i)=b(i+1)-b(i)-(h(i)*v(i-1))/u(i-1);
    end
    z(n1)=0;
    for i=(n1-1:-1:2)
        z(i)=(v(i-1)-(h(i)*z(i+1)))/u(i-1);
    end
    z(1)=0;
    %Encontramos el nuevo punto para determinar a que polinomio pertenece.
    for i=(1:n2)
        distancia=abs(min(x)-max(x));
        for j=(1:n1)
            if(abs(newx(i)-x(j))<distancia)
                distancia=abs(newx(i)-x(j));
                indx=j;
            end
        end
        %Aproximamos los nuevos puntos y.
        if(newx(i)<x(indx))
            indx=indx-1;
            A=(1/(6*h(indx)))*(z(indx+1)-z(indx));
            B=z(indx)/2;
            C=(-h(indx)/6)*(z(indx+1))-(h(indx)/3)*z(indx)+(1/h(indx))*(y(indx
            +1)-y(indx));
            newy(i)=y(indx)+(newx(i)-x(indx))*(C+(newx(i)-x(indx))*(B+(newx(i
            )-x(indx))*A));
        elseif(newx(i)>x(indx))
            A=(1/(6*h(indx)))*(z(indx+1)-z(indx));
            B=z(indx)/2;
            C=(-h(indx)/6)*(z(indx+1))-(h(indx)/3)*z(indx)+(1/h(indx))*(y(indx
            +1)-y(indx));
```

```
61              newy(i)=y(indx)+(newx(i)-x(indx))*(C+(newx(i)-x(indx))*(B+(newx(i
                )-x(indx))*A));
62          elseif((newx(i)-x(indx))<10^-8)
63              newy(i)=y(indx);
64          end
65      end
66  end
67
68  %Parámetros de Salida:
69
70  %newy:Nuevos puntos estimados mediante Spline Natural.
71
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                        %
%Interpolación Spline Cúbico Completo%
%                                        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Parámetros de Entrada:

%x,y:Vectores que guardan los números de los pares ordenado (xi,yi)
%    para i=0,1,2,....,n donde x0<x1<x2<...<xn.

%newx:Vector de elementos para los cuales se estimarán mediante la
Interpolación
%     Spline Cúbico Completo.

function[newy]=SCCompleto(x,y,dfa,dfb,newx)
    n1=length(x);%Obtenemos el número de elementos de x.
    n2=length(newx);%Obtenemos el número de elementos de newx.
    newy=zeros(n2,1);%Vector que guardará los nuevos valores de y.

    %Calcuamos z para encontrar los coeficientes del polinomio.
    h=zeros(n1-1,1);
    a=zeros(n1,1);
    l=zeros(n1,1);
    u=zeros(n1,1);
    zp=zeros(n1,1);
    z=zeros(n1,1);
    for i=(1:n1-1)
        h(i)=x(i+1)-x(i);
    end
    a(1)=(3*(y(2)-y(1)))/h(1)-3*dfa;
    a(n1)=3*dfb-3*((a(n1)-a(n1-1))/h(n1-1));
    for i=(2:n1-1)
        a(i)=(3/h(i))*(y(i+1)-y(i))-(3/h(i-1))*(y(i)-y(i-1));
    end
    l(1)=2*h(1);
    u(1)=0.5;
    zp(1)=a(1)/l(1);
    for i=(2:n1-1)
        l(i)=2*(x(i+1)-x(i-1))-h(i-1)*u(i-1);
        u(i)=h(i)/l(i);
        zp(i)=(a(i)-h(i-1)*zp(i-1))/l(i);
    end
    l(n1)=h(n1-1)*(2-u(n1-1));
    zp(n1)=(a(n1)-h(n1-1)*zp(n1-1))/l(n1);
    z(n1)=zp(n1);
    for i=(n1-1:-1:1)
        z(i)=zp(i)-u(i)*z(i+1);
    end
    %Encontramos el nuevo punto para determinar a que polinomio pertenece.
    for i=(1:n2)
        distancia=abs(min(x)-max(x));
        for j=(1:n1)
            if(abs(newx(i)-x(j))<distancia)
                distancia=abs(newx(i)-x(j));
                indx=j;
            end
        end
        %Aproximamos los nuevos puntos y.
        if(newx(i)<x(indx))
            indx=indx-1;
            A=(1/(6*h(indx)))*(z(indx+1)-z(indx));
            B=z(indx)/2;
            C=(-h(indx)/6)*(z(indx+1))-(h(indx)/3)*z(indx)+(1/h(indx))*(y(indx
```

```
            +1)-y(indx));
64              newy(i)=y(indx)+(newx(i)-x(indx))*(C+(newx(i)-x(indx))*(B+(newx(i
                )-x(indx))*A));
65          elseif(newx(i)>x(indx))
66              A=(1/(6*h(indx)))*(z(indx+1)-z(indx));
67              B=z(indx)/2;
68              C=(-h(indx)/6)*(z(indx+1))-(h(indx)/3)*z(indx)+(1/h(indx))*(y(indx
                +1)-y(indx));
69              newy(i)=y(indx)+(newx(i)-x(indx))*(C+(newx(i)-x(indx))*(B+(newx(i
                )-x(indx))*A));
70          elseif((newx(i)-x(indx))<10^-8)
71              newy(i)=y(indx);
72          end
73      end
74
75  end
76
77  %Parámetros de Salida:
78
79  %newy:Nuevos puntos estimados mediante Spline Completo.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                       %
%Interpolación Spline Cúbico Knot-A-Not %
%                                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Parámetros de Entrada:

%x,y:Vectores que guardan los números de los pares ordenado (xi,yi)
%    para i=0,1,2,....,n donde x0<x1<x2<...<xn.

%newx:Vector de elementos para los cuales se estimarán mediante la Interpolación
%     Spline Cúbico Knot-A-Not.

function[newy]=SCNKnot(x,y,newx)
    n1=length(x);%Obtenemos el número de elementos de x.
    n2=length(newx);%Obtenemos el número de elementos de newx.
    newy=zeros(n2,1);%Vector que guardará los nuevos valores de y.

    %Calcuamos los coeficientes de los polinomio.
    z=zeros(n1,1);
    h=zeros(n1-1,1);
    b=zeros(n1,1);
    for i=(1:n1-1)
        h(i)=x(i+1)-x(i);
        if(i>2)
            b(i)=((6/h(i))*(y(i+1)-y(i)))-((6/h(i-1))*(y(i)-y(i-1)));
        end
    end
    e=zeros(n1-2,1);
    a=zeros(n1-1,1);
    d=zeros(n1,1);
    c=zeros(n1-1,1);
    f=zeros(n1-2,1);
    e(n1-2)=-h(n1-1);
    a(n1-1)=h(n1-2)+h(n1-1);
    d(n1)=-h(n1-2);
    d(1)=-h(2);
    c(1)=h(1)+h(2);
    f(1)=-h(1);
    for i=(2:n1-1)
        a(i-1)=h(i-1);
        d(i)=2*(h(i)+h(i-1));
        c(i)=h(i);
    end
    b(1)=0;
    b(n1)=0;

    r=a(1);
    s=a(2);
    t=e(1);
    for i=(2:n1-1)
        zmult=r/d(i-1);
        d(i)=d(i)-zmult*c(i-1);
        c(i)=c(i)-zmult*f(i-1);
        b(i)=b(i)-zmult*b(i-1);
        zmult=t/d(i-1);
        r=s-zmult*c(i-1);
        d(i+1)=d(i+1)-zmult*f(i-1);
        b(i+1)=b(i+1)-zmult*b(i-1);
        if(i<n1-2)
            s=a(i+1);
            t=e(i);
```

```
64          end
65      end
66      zmult=r/d(n1-1);
67      d(n1)=d(n1)-zmult*c(n1-1);
68      z(n1)=(b(n1)-zmult*b(n1-1))/d(n1);
69      z(n1-1)=(b(n1-1)-c(n1-1)*z(n1))/d(n1-1);
70      for i=(n1-2:-1:1)
71          z(i)=(b(i)-f(i)*z(i+2)-c(i)*z(i+1))/d(i);
72      end
73      %Encontramos el nuevo punto para determinar a que polinomio pertenece.
74      for i=(1:n2)
75          distancia=abs(min(x)-max(x));
76          for j=(1:n1)
77              if(abs(newx(i)-x(j))<distancia)
78                  distancia=abs(newx(i)-x(j));
79                  indx=j;
80              end
81          end
82          %Aproximamos los nuevos puntos y.
83          if(newx(i)<x(indx))
84              indx=indx-1;
85              A=(1/(6*h(indx)))*(z(indx+1)-z(indx));
86              B=z(indx)/2;
87              C=(-h(indx)/6)*(z(indx+1))-(h(indx)/3)*z(indx)+(1/h(indx))*(y(indx
                +1)-y(indx));
88              newy(i)=y(indx)+(newx(i)-x(indx))*(C+(newx(i)-x(indx))*(B+(newx(i
                )-x(indx))*A));
89          elseif(newx(i)>x(indx))
90              A=(1/(6*h(indx)))*(z(indx+1)-z(indx));
91              B=z(indx)/2;
92              C=(-h(indx)/6)*(z(indx+1))-(h(indx)/3)*z(indx)+(1/h(indx))*(y(indx
                +1)-y(indx));
93              newy(i)=y(indx)+(newx(i)-x(indx))*(C+(newx(i)-x(indx))*(B+(newx(i
                )-x(indx))*A));
94          elseif((newx(i)-x(indx))<10^-8)
95              newy(i)=y(indx);
96          end
97      end
98  end
99
100 %Parámetros de Salida:
101
102 %newy:Nuevos puntos estimados mediante Spline Cúbico Knot-A-Not.
103
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                  %
%Interpolación Cúbica de Akima     %
%                                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Parámetros de Entrada:

%x,y:Vectores que guardan los números de los pares ordenado (xi,yi)
%    para i=0,1,2,....,n donde x0<x1<x2<...<xn.

%newx:Vector de elementos para los cuales se estimarán mediante la
Interpolación
%     Cúbica de Akima.

function[newy]=ICAkima(x,y,newx)
    n1=length(x);%Obtenemos el número de elementos de x.
    n2=length(newx);%Obtenemos el número de elementos de newx.
    newy=zeros(n2,1);%Vector que guardará los nuevos valores de y.
    %Vectores necesarios para calcular los coeficientes de los polinomios.
    xadd=zeros(n1+4,1);
    yadd=zeros(n1+4,1);
    h=zeros(n1-1,1);
    s=zeros(n1+3,1);
    dy=zeros(n1,1);
    a=zeros(n1-1,1);
    b=zeros(n1-1,1);
    c=zeros(n1-1,1);
    d=zeros(n1-1,1);

    %Calculamos los nuevos puntos extra necesarios.
    xadd(n1+4)=2*x(n1)-x(n1-2);
    xadd(n1+3)=x(n1)+x(n1-1)-x(n1-2);
    xadd(1)=2*x(1)-x(3);
    xadd(2)=x(1)+x(2)-x(3);
    yadd(n1+3)=((((2*(y(n1)-y(n1-1)))/(x(n1)-x(n1-1)))-((y(n1-1)-y(n1-2))/(x(
    n1-1)-x(n1-2))))*(xadd(n1+3)-x(n1)))+y(n1);
    yadd(n1+4)=((((2*(yadd(n1+3)-y(n1)))/(xadd(n1+3)-x(n1)))-((y(n1)-y(n1-1
    ))/(x(n1)-x(n1-1))))*(xadd(n1+4)-xadd(n1+3)))+yadd(n1+3);
    yadd(2)=-((((2*(y(2)-y(1)))/(x(2)-x(1)))-((y(3)-y(2))/(x(3)-x(2))))*(x(1)-
    xadd(2)))+y(1);
    yadd(1)=-((((2*(y(1)-yadd(2)))/(x(1)-xadd(2)))-((y(2)-y(1))/(x(2)-x(1
    ))))*(xadd(2)-xadd(1)))+yadd(2);
    for i=(1:n1-1)
        xadd(i+2)=x(i);
        yadd(i+2)=y(i);
        h(i)=x(i+1)-x(i);
    end
    xadd(n1+2)=x(n1);
    yadd(n1+2)=y(n1);

    %Calculamos los coeficientes del polinomio.
    for i=(1:n1+3)
        s(i)=(yadd(i+1)-yadd(i))/(xadd(i+1)-xadd(i));
    end
    for i=(3:n1+2)
        w1=abs(s(i+1)-s(i));
        w2=abs(s(i-1)-s(i-2));
        if(w1<=10^-8 && w2<=10^-8)
            dy(i-2)=(s(i-1)+s(i))/2;
        else
            dy(i-2)=(w1*s(i-1)+w2*s(i))/(w1+w2);
        end
    end
```

```
60        for i=(1:n1-1)
61            a(i)=y(i);
62            b(i)=dy(i);
63            c(i)=(3*s(i+2)-dy(i+1)-2*dy(i))/(h(i));
64            d(i)=-(2*s(i+2)-dy(i+1)-dy(i))/(h(i)^2);
65        end
66        %Encontramos el nuevo punto para determinar a que polinomio pertenece.
67        for i=(1:n2)
68            distancia=abs(min(x)-max(x));
69            for j=(1:n1)
70                if(abs(newx(i)-x(j))<distancia)
71                    distancia=abs(newx(i)-x(j));
72                    indx=j;
73                end
74            end
75            %Aproximamos los nuevos puntos y.
76            if(newx(i)<x(indx))
77                indx=indx-1;
78                newy(i)=a(indx)+b(indx)*(newx(i)-x(indx))+c(indx)*(newx(i)-x(indx
                  ))^2+d(indx)*(newx(i)-x(indx))^3;
79            elseif(newx(i)>x(indx))
80                newy(i)=a(indx)+b(indx)*(newx(i)-x(indx))+c(indx)*(newx(i)-x(indx
                  ))^2+d(indx)*(newx(i)-x(indx))^3;
81            elseif((newx(i)-x(indx))<10^-8)
82                newy(i)=y(indx);
83            end
84        end
85
86    end
87
88    %Parámetros de Salida:
89
90    %newy:Nuevos puntos estimados mediante Interpolación de Akima.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                            %
%Interpolación Cúbica de Restringida         %
%                                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Parámetros de Entrada:

%x,y:Vectores que guardan los números de los pares ordenado (xi,yi)
%    para i=0,1,2,....,n donde x0<x1<x2<...<xn.

%newx:Vector de elementos para los cuales se estimarán mediante la
Interpolación
%     Cúbica Restringida.

function[newy]=ICCons(x,y,newx)
    n1=length(x);%Obtenemos el número de elementos de x.
    n2=length(newx);%Obtenemos el número de elementos de newx.
    newy=zeros(n2,1);%Vector que guardará los nuevos valores de y.
    %Vectores necesarios para calcular los coeficientes de los polinomios.
    h=zeros(n1-1,1);
    s=zeros(n1+3,1);
    dy=zeros(n1,1);
    a=zeros(n1-1,1);
    b=zeros(n1-1,1);
    c=zeros(n1-1,1);
    d=zeros(n1-1,1);
    %Calcuamos los coeficientes de los polinomio.
    for i=(1:n1-1)
        s(i)=(y(i+1)-y(i))/(x(i+1)-x(i)) ;
        h(i)=x(i+1)-x(i);
    end
    for i=(2:n1-1)
        if(sign(s(i-1))~=sign(s(i)))
            dy(i)=0;
        else
            dy(i)=2/((1/s(i))+(1/s(i-1)));
        end
    end
    dy(1)=((3/2)*s(1))-((1/2)*dy(2));
    dy(n1)=((3/2)*s(n1-1))-((1/2)*dy(n1-1));
    for i=(1:n1-1)
        a(i)=y(i);
        b(i)=dy(i);
        c(i)=(3*s(i)-dy(i+1)-2*dy(i))/(h(i));
        d(i)=-(2*s(i)-dy(i+1)-dy(i))/(h(i)^2);
    end
    %Encontramos el nuevo punto para determinar a que polinomio pertenece.
    for i=(1:n2)
        distancia=abs(min(x)-max(x));
        for j=(1:n1)
            if(abs(newx(i)-x(j))<distancia)
                distancia=abs(newx(i)-x(j));
                indx=j;
            end
        end
        %Aproximamos los nuevos puntos y.
        if(newx(i)<x(indx))
            indx=indx-1;
            newy(i)=a(indx)+b(indx)*(newx(i)-x(indx))+c(indx)*(newx(i)-x(indx
            ))^2+d(indx)*(newx(i)-x(indx))^3;
        elseif(newx(i)>x(indx))
            newy(i)=a(indx)+b(indx)*(newx(i)-x(indx))+c(indx)*(newx(i)-x(indx
            ))^2+d(indx)*(newx(i)-x(indx))^3;
```

```
62          elseif((newx(i)-x(indx))<10^-8)
63              newy(i)=y(indx);
64          end
65      end
66  end
67
68  %Parámetros de Salida:
69
70  %newy:Nuevos puntos estimados mediante Interpolación Restringida.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                  %
%Interpolación Cúbica Monótona     %
%                                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Parámetros de Entrada:

%x,y:Vectores que guardan los números de los pares ordenado (xi,yi)
%    para i=0,1,2,....,n donde x0<x1<x2<...<xn.

%newx:Vector de elementos para los cuales se estimarán mediante la
Interpolación
%     Cúbica Monótona.

function[newy]=MPIH(x,y,newx)
    n1=length(x);%Obtenemos el número de elementos de x.
    n2=length(newx);%Obtenemos el número de elementos de newx.
    newy=zeros(n2,1);%Vector que guardará los nuevos valores de y.
    %Vectores necesarios para calcular los coeficientes de los polinomios.
    h=zeros(n1-1,1);
    s=zeros(n1+3,1);
    dy=zeros(n1,1);
    a=zeros(n1-1,1);
    b=zeros(n1-1,1);
    c=zeros(n1-1,1);
    d=zeros(n1-1,1);

    %Calculamos los coeficientes del polinomio.
    for i=(1:n1-1)
        s(i)=(y(i+1)-y(i))/(x(i+1)-x(i));
        h(i)=x(i+1)-x(i);
    end
    for i=(2:n1-1)
        if ((s(i-1)*s(i))<=0)
            dy(i)=0;
        else
            w1=((h(i-1)+2*h(i))/(3*(h(i)+h(i-1))))*(1/s(i-1));
            w2=((2*h(i-1)+h(i))/(3*(h(i)+h(i-1))))*(1/s(i));
            dy(i)=1/(w1+w2);
        end
    end
    g1=(((2*h(1)+h(2))/(h(1)+h(2)))*s(1))-((h(1)/(h(1)+h(2)))*s(2));
    gn1=(((2*h(n1-1)+h(n1-2))/(h(n1-1)+h(n1-2)))*s(n1-1))-((h(n1-1)/(h(n1-1)+h
    (n1-2)))*s(n1-2));
    if(g1*s(1)<=0)
        dy(1)=0;
    elseif(s(1)*s(2)<=0 && abs(g1)>3*abs(s(1)))
        dy(1)=3*s(1);
    else
        dy(1)=g1;
    end
    if(gn1*s(n1-1)<=0)
        dy(n1)=0;
    elseif(s(n1-1)*s(n1-2)<=0 && abs(gn1)>3*abs(s(n1-1)))
        dy(n1)=3*s(n1-1);
    else
        dy(n1)=gn1;
    end
    for i=(1:n1-1)
        a(i)=y(i);
        b(i)=dy(i);
        c(i)=(3*s(i)-dy(i+1)-2*dy(i))/(h(i));
        d(i)=-(2*s(i)-dy(i+1)-dy(i))/(h(i)^2);
```

```
63      end
64      %Encontramos el nuevo punto para determinar a que polinomio pertenece.
65      for i=(1:n2)
66          distancia=abs(min(x)-max(x));
67          for j=(1:n1)
68              if(abs(newx(i)-x(j))<distancia)
69                  distancia=abs(newx(i)-x(j));
70                  indx=j;
71              end
72          end
73          %Aproximamos los nuevos puntos y.
74          if(newx(i)<x(indx))
75              indx=indx-1;
76              newy(i)=a(indx)+b(indx)*(newx(i)-x(indx))+c(indx)*(newx(i)-x(indx
                ))^2+d(indx)*(newx(i)-x(indx))^3;
77          elseif(newx(i)>x(indx))
78              newy(i)=a(indx)+b(indx)*(newx(i)-x(indx))+c(indx)*(newx(i)-x(indx
                ))^2+d(indx)*(newx(i)-x(indx))^3;
79          elseif((newx(i)-x(indx))<10^-8)
80              newy(i)=y(indx);
81          end
82      end
83  end
84  %Parámetros de Salida:
85
86  %newy:Nuevos puntos estimados mediante Interpolación Cúbica Monótona.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                %
%Función que encuentra los puntos cercanos       %
%                                                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%Parámetros de Entrada:

%xd,yd:Vectores que guardan los números de (xi,yi).

%ncp: Número de puntos cercanos requeridos (mínimo 3).

function[ipc]=idcldp (xd, yd, ncp)

    %Función para calcular distancias.

    function[dis]= dsqf(x1,y1,x2,y2)
        dis=(x2-x1)^2+(y2-y1)^2;
    end

    %Proceso preeliminar.

    ndp=length(xd);
    ipc=zeros(ndp*ncp,1);
    dis0=zeros(ndp,1);
    iip0=0;

    %Obtenemos los puntos cercanos.

    for i=(1:ndp)
        if(i<ndp)
            for j=(i+1:ndp)
                dis0(j)= dsqf(xd(i),yd(i),xd(j),yd(j));
            end
        end
        if(i>1)
            for j=(1:i-1)
                dis0(j)= dsqf(xd(i),yd(i),xd(j),yd(j));
            end
        end
        maxim=max(dis0)+1;
        dis0(i)=maxim;
        for j=(1:ncp)
            iip0=iip0+1;
            [~,ipc0]=min(dis0);
            ipc(iip0)=ipc0;
            dis0(ipc0)=maxim;
        end
        dis0=zeros(ndp,1);
    end
end

%Parámetros de Salida:

%ipc: Índice de los puintos cercanos.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                      %
%Función que aproxima las derivadas parciales          %
%                                                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Parámetros de Entrada:

%xd,yd,zd:Vectores que guardan los números de (xi,yi,zi).

%ncp: Número de puntos cercanos requeridos (mínimo 3).

%ipc: Índice de puntos cercanos.

function[pd]=idpdrv(xd,yd,zd,ncp,ipc)

    %Proceso preeliminar.

    ndp=length(xd);
    pd=zeros(5*ndp,1);
    ndp0=ndp;
    ncp0=ncp;
    ncpm1=ncp0-1;

    %Estimamos las derivadas parciales ZX y ZY.

    for ip0=(1:ndp0)
        x0 = xd(ip0);
        y0 = yd(ip0);
        z0 = zd(ip0);
        nmx = 0;
        nmy = 0;
        nmz = 0;
        jipc0 = ncp0*(ip0-1);
        for ic1=(1:ncpm1)
            jipc = jipc0+ic1;
            ipi = ipc(jipc);
            dx1 = xd(ipi)-x0;
            dy1 = yd(ipi)-y0;
            dz1 = zd(ipi)-z0;
            ic2mn = ic1+1;
            for ic2=(ic2mn:ncp0)
                jipc = jipc0+ic2;
                ipi = ipc(jipc);
                dx2 = xd(ipi)-x0;
                dy2 = yd(ipi)-y0;
                dnmz = dx1*dy2-dy1*dx2;
                if (dnmz== 0)
                    continue;
                end
                dz2 = zd(ipi)-z0;
                dnmx = dy1*dz2-dz1*dy2;
                dnmy = dz1*dx2-dx1*dz2;
                if (dnmz < 0)
                    dnmx =-dnmx;
                    dnmy =-dnmy;
                    dnmz =-dnmz;
                end
                nmx = nmx+dnmx;
                nmy = nmy+dnmy;
                nmz = nmz+dnmz;
            end
        end
        jpd0 = 5*ip0;
```

```
        pd(jpd0-4) = -nmx/nmz;
        pd(jpd0-3) = -nmy/nmz;
    end

    %Estimamos las derivadas parciales ZXX, ZXY, y ZYY.

    for ip0 =(1:ndp0)
        x0 = xd(ip0);
        jpd0 = 5*ip0;
        y0 = yd(ip0);
        zx0 = pd(jpd0-4);
        zy0 = pd(jpd0-3);
        nmxx = 0;
        nmxy = 0;
        nmyx = 0;
        nmyy = 0;
        nmz  = 0;
        jipc0 = ncp0*(ip0-1);
        for ic1 =(1:ncpm1)
            jipc = jipc0+ic1;
            ipi = ipc(jipc);
            dx1 = xd(ipi)-x0;
            dy1 = yd(ipi)-y0;
            jpd = 5*ipi;
            dzx1 = pd(jpd-4)-zx0;
            dzy1 = pd(jpd-3)-zy0;
            ic2mn = ic1+1;
            for ic2 =(ic2mn:ncp0)
                jipc = jipc0+ic2;
                ipi = ipc(jipc);
                dx2 = xd(ipi)-x0;
                dy2 = yd(ipi)-y0;
                dnmz  = dx1*dy2 -dy1*dx2;
                if ( dnmz == 0 )
                    continue;
                end
                jpd = 5 * ipi;
                dzx2 = pd(jpd-4) - zx0;
                dzy2 = pd(jpd-3) - zy0;
                dnmxx = dy1 * dzx2 - dzx1 * dy2;
                dnmxy = dzx1 * dx2 - dx1 * dzx2;
                dnmyx = dy1 * dzy2 - dzy1 * dy2;
                dnmyy = dzy1 * dx2 - dx1 * dzy2;
                if ( dnmz < 0)
                    dnmxx = -dnmxx;
                    dnmxy = -dnmxy;
                    dnmyx = -dnmyx;
                    dnmyy = -dnmyy;
                    dnmz  = -dnmz;
                end
                nmxx = nmxx+dnmxx;
                nmxy = nmxy+dnmxy;
                nmyx = nmyx+dnmyx;
                nmyy = nmyy+dnmyy;
                nmz  = nmz +dnmz;
            end
        end
        pd(jpd0-2) = - nmxx / nmz;
        pd(jpd0-1) = - ( nmxy + nmyx ) / ( 2 * nmz );
        pd(jpd0)   = - nmyy / nmz;
    end
end

%Parámetros de Salida:
```

```
129
130   %pd: Aproximación a las derivadas parciales ZX,ZY,ZXX,ZXY,ZYY.
131
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                          %
%   Función que crea una malla triangular  %
%                                          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%Parámetros de Entrada:

%xd,yd:Vectores que guardan los números de (xi,yi).

function[nt, ipt, nl, ipl, iwl, iwp, wk]=idtang(xd,yd)

    %Funciones para cálculos de la malla.

    function[d]=dsqf(u1,v1,u2,v2)
        d=(u2-u1)^2+(v2-v1)^2;
    end
    function[lado]=side(u1,v1,u2,v2,u3,v3)
        lado=(v3-v1)*(u2-u1)-(u3-u1)*(v2-v1);
    end

    %Proceso preeliminar.

    ratio=10^(-6);
    nrep = 100;
    itf=zeros(2,1);
    ndp=length(xd);
    ipt=zeros(6*ndp-15,1);
    ipl=zeros(6*ndp,1);
    iwl=zeros(18*ndp,1);
    iwp=zeros(ndp,1);
    wk=zeros(ndp,1);
    ndp0=ndp;
    ndpm1=ndp0-1;

    %Determinamos el par de puntos más cercanos y el punto medio.

    dsqmn = dsqf(xd(1),yd(1),xd(2),yd(2));
    ipmn1 = 1;
    ipmn2 = 2;
    for ip1=(1:ndpm1)
        x1 = xd(ip1);
        y1 = yd(ip1);
        ip1p1 = ip1+1;
        for ip2=(ip1p1:ndp0)
            dsqi = dsqf(x1,y1,xd(ip2),yd(ip2));
            if ( dsqi < dsqmn )
                dsqmn = dsqi;
                ipmn1 = ip1;
                ipmn2 = ip2;
            end
        end
    end
    dsq12 = dsqmn;
    xdmp = (xd(ipmn1)+xd(ipmn2))/2;
    ydmp = (yd(ipmn1)+yd(ipmn2))/2;

    %Ordenamos los ndp-2 puntos restantes de forma ascendente con
    %respecto a la distancia del punto medio
    %calculado anteriormente.

    jp1 = 2;
    for ip1 = (1:ndp0)
```

```
65          if ( ip1 ~= ipmn1 && ip1 ~= ipmn2 )
66              jp1 = jp1+1;
67              iwp(jp1) = ip1;
68              wk(jp1) = dsqf(xdmp,ydmp,xd(ip1),yd(ip1));
69          end
70      end
71      for jp1 = (3:ndpm1)
72          dsqmn = wk(jp1);
73          jpmn = jp1;
74          for jp2 = (jp1:ndp0)
75              if ( wk(jp2) < dsqmn )
76                  dsqmn = wk(jp2);
77                  jpmn = jp2;
78              end
79          end
80          its = iwp(jp1);
81          iwp(jp1) = iwp(jpmn);
82          iwp(jpmn) = its;
83          wk(jpmn) = wk(jp1);
84      end
85
86      %Modificamos los puntos de tal manera que los primeros tres puntos
87      %no sean colineales.
88
89      ar = dsq12*ratio;
90      x1 = xd(ipmn1);
91      y1 = yd(ipmn1);
92      dx21 = xd(ipmn2)-x1;
93      dy21 = yd(ipmn2)-y1;
94      for jp=(3:ndp0)
95          ip = iwp(jp);
96          if(abs((yd(ip)-y1)*dx21-(xd(ip)-x1)*dy21)>ar)
97              break;
98          end
99      end
100
101     if ( jp ~= 3 )
102         jpmx = jp;
103         jp = jpmx+1;
104         for jpc =(4:jpmx)
105             jp = jp-1;
106             iwp(jp) = iwp(jp-1);
107         end
108         iwp(3) = ip;
109     end
110
111     %Para el primer triangulo guardamos
112     %en número de vértices en ipt y los
113     %bordes en ipl.
114
115     ip1 = ipmn1;
116     ip2 = ipmn2;
117     ip3 = iwp(3);
118     if(side(xd(ip1),yd(ip1),xd(ip2),yd(ip2),xd(ip3),yd(ip3))<0)
119         ip1 = ipmn2;
120         ip2 = ipmn1;
121     end
122
123     nt0 = 1;
124     ntt3 = 3;
125     ipt(1) = ip1;
126     ipt(2) = ip2;
127     ipt(3) = ip3;
128     nl0 = 3;
```

```
129       nlt3 = 9;
130       ipl(1) = floor(ip1);
131       ipl(2) = floor(ip2);
132       ipl(3) = floor(1);
133       ipl(4) = floor(ip2);
134       ipl(5) = floor(ip3);
135       ipl(6) = floor(1);
136       ipl(7) = floor(ip3);
137       ipl(8) = floor(ip1);
138       ipl(9) = 1;
139
140       %Seguimos el proceso con los ndp-3
141       %puntos restantes.
142
143       for jp1=(4:ndp0)
144           ip1 = iwp(jp1);
145           x1 = xd(ip1);
146           y1 = yd(ip1);
147
148           %Determinamos los bordes de los
149           %segmentos de linea.
150
151           ip2 = ipl(1);
152           jpmn = 1;
153           dxmn = xd(ip2)-x1;
154           dymn = yd(ip2)-y1;
155           dsqmn = dxmn^2+dymn^2;
156           armn = dsqmn*ratio;
157           jpmx = 1;
158           dxmx = dxmn;
159           dymx = dymn;
160           dsqmx = dsqmn;
161           armx = armn;
162           for jp2=(2:nl0)
163               ip2 = ipl(3*jp2-2);
164               dx = xd(ip2)-x1;
165               dy = yd(ip2)-y1;
166               ar = dy*dxmn-dx*dymn;
167              if(ar<=armn)
168                   dsqi = dx^2+dy^2;
169                   if(~(ar>=(-armn)&& dsqi>=dsqmn))
170                       jpmn = jp2;
171                       dxmn = dx;
172                       dymn = dy;
173                       dsqmn = dsqi;
174                       armn = dsqmn*ratio;
175                   end
176               end
177
178               ar = dy*dxmx-dx*dymx;
179               if(ar>=(-armx))
180                   dsqi = dx^2+dy^2;
181                   if(~(ar<=armx && dsqi>=dsqmx))
182                       jpmx = jp2;
183                       dxmx = dx;
184                       dymx = dy;
185                       dsqmx = dsqi;
186                       armx = dsqmx*ratio;
187                   end
188                end
189
190           end
191           if(jpmx<jpmn)
192               jpmx = jpmx+nl0;
```

```
193         end
194         nsh = jpmn-1;
195
196         %Seguimos obteniento los nuevos bordes para guardar en
197         %ipl.
198
199         if(nsh>0)
200             nsht3 = nsh*3;
201             for jp2t3 = (3:3:nsht3)
202                 jp3t3 = jp2t3+nlt3;
203                 ipl(jp3t3-2) = ipl(jp2t3-2);
204                 ipl(jp3t3-1) = ipl(jp2t3-1);
205                 ipl(jp3t3)   = ipl(jp2t3);
206             end
207             for jp2t3 = (3:3:nlt3)
208                 jp3t3 = jp2t3+nsht3;
209                 ipl(jp2t3-2) = ipl(jp3t3-2);
210                 ipl(jp2t3-1) = ipl(jp3t3-1);
211                 ipl(jp2t3)   = ipl(jp3t3);
212             end
213             jpmx = jpmx-nsh;
214         end
215
216         %Seguimos creando más triangulos
217         %y actualizamos los bordes de
218         %cada triangulo usando ipt,ipl,iwl.
219
220         jwl=0;
221         for jp2 = (jpmx:nl0)
222             jp2t3 = jp2*3;
223             ipl1 = ipl(jp2t3-2);
224             ipl2 = ipl(jp2t3-1);
225             it = ipl(jp2t3);
226
227             %Tomamos los triangulos de ipt.
228
229             nt0 = nt0+1;
230             ntt3 = ntt3+3;
231             ipt(ntt3-2) = ipl2;
232             ipt(ntt3-1) = ipl1;
233             ipt(ntt3)   = ip1;
234
235             %Actualizamos los bordes.
236
237             if(jp2==jpmx)
238                 ipl(jp2t3-1) = ip1;
239                 ipl(jp2t3)= nt0;
240             end
241             %61
242             if(jp2==nl0)
243                 nln = jpmx+1;
244                 nlnt3 = nln*3;
245                 ipl(nlnt3-2) = ip1;
246                 ipl(nlnt3-1) = ipl(1);
247                 ipl(nlnt3)   = nt0;
248             end
249
250             %Determinamos los vértices que
251             %no están sobre el borde.
252
253             itt3 = it*3;
254             ipti = ipt(itt3-2);
255             if(~(ipti~=ipl1 && ipti~=ipl2))
256                 ipti = ipt(itt3-1);
```

```
257                 if(~(ipti~=ipl1&&ipti~=ipl2))
258                     ipti = ipt(itt3);
259                 end
260             end
261
262             %Revisamos el intercambio si es necesario.
263
264             if(idxchg(xd,yd,ip1,ipti,ipl1,ipl2)~=0)
265
266                 %Modificamos ipt cuando sea necesario.
267                 ipt(itt3-2) = ipti;
268                 ipt(itt3-1) = ipl1;
269                 ipt(itt3)   = ip1;
270                 ipt(ntt3-1) = ipti;
271                 if(jp2==jpmx)
272                     ipl(jp2t3) = it;
273                 end
274                 if(jp2==nl0&&ipl(3)==it)
275                     ipl(3) = nt0;
276                 end
277
278                 %Usamos iwl.
279
280                 jwl = jwl+4;
281                 iwl(jwl-3) = ipl1;
282                 iwl(jwl-2) = ipti;
283                 iwl(jwl-1) = ipti;
284                 iwl(jwl)   = ipl2;
285             end
286
287         end
288         nl0 = nln;
289         nlt3 = nlnt3;
290         nlf = jwl/2;
291
292         %Continua la triangulación.
293
294         if(nlf~=0)
295
296             ntt3p3 = ntt3+3;
297             for irep = (1:nrep)
298                 for ilf = (1:nlf)
299                     ilft2 = ilf*2;
300                     ipl1 = iwl(ilft2-1);
301                     ipl2 = iwl(ilft2);
302
303                     %Buscamos en ipt dos lados que que coincidan.
304
305                     ntf=0;
306                     for itt3r =(3:3:ntt3)
307                         itt3 = ntt3p3-itt3r;
308                         ipt1 = ipt(itt3-2);
309                         ipt2 = ipt(itt3-1);
310                         ipt3 = ipt(itt3);
311                         if(~(ipl1~=ipt1&&ipl1~=ipt2&&ipl1~=ipt3))
312                             if(~(ipl2~=ipt1&&ipl2~=ipt2 &&ipl2~=ipt3))
313                                 ntf = ntf+1;
314                                 itf(ntf) = itt3/3;
315                                 if(ntf==2)
316                                     break;
317                                 end
318                             end
319                         end
320
```

```
321                     end
322
323                     %Determinamos los vértices del triangulo.
324
325                     if(ntf>=2)
326
327                         it1t3 = itf(1)*3;
328                         ipti1 = ipt(it1t3-2);
329                         if(~(ipti1~=ipl1&&ipti1~=ipl2))
330                             ipti1 = ipt(it1t3-1);
331                             if(~(ipti1~=ipl1&&ipti1~=ipl2))
332                                 ipti1 = ipt(it1t3);
333                             end
334                         end
335
336                         it2t3 = itf(2)*3;
337                         ipti2 = ipt(it2t3-2);
338                         if(~(ipti2~=ipl1&&ipti2~=ipl2))
339                             ipti2 = ipt(it2t3-1);
340                             if(~(ipti2~=ipl1&&ipti2~=ipl2))
341                                 ipti2 = ipt(it2t3);
342                             end
343                         end
344
345                         %Revisamos nuevamente.
346
347                          if(~(idxchg(xd,yd,ipti1,ipti2,ipl1,ipl2)==0))
348
349                             %Modificamos ipt si es necesario.
350
351                             ipt(it1t3-2) = ipti1;
352                             ipt(it1t3-1) = ipti2;
353                             ipt(it1t3)   = ipl1;
354                             ipt(it2t3-2) = ipti2;
355                             ipt(it2t3-1) = ipti1;
356                             ipt(it2t3)   = ipl2;
357
358                             %Creamos nuevos marcadores.
359
360                             jwl = jwl+8;
361                             iwl(jwl-7) = ipl1;
362                             iwl(jwl-6) = ipti1;
363                             iwl(jwl-5) = ipti1;
364                             iwl(jwl-4) = ipl2;
365                             iwl(jwl-3) = ipl2;
366                             iwl(jwl-2) = ipti2;
367                             iwl(jwl-1) = ipti2;
368                             iwl(jwl)   = ipl1;
369                             for jlt3 = (3:3:nlt3)
370                                 iplj1 = ipl(jlt3-2);
371                                 iplj2 = ipl(jlt3-1);
372                                 if((iplj1==ipl1&&iplj2==ipti2)|||(iplj2==ipl1&&
                                    iplj1==ipti2))
373                                     ipl(jlt3) = itf(1);
374                                 end
375                                 if((iplj1==ipl2&&iplj2==ipti1)|||(iplj2==ipl2&&
                                    iplj1==ipti1))
376                                     ipl(jlt3) = itf(2);
377                                  end
378
379                             end
380
381                          end
382                     end
```

```
                end
                nlfc = nlf;
                nlf = jwl/2;
                if(nlf~=nlfc)

                    %Reiniciamos iwl.

                    jwl = 0;
                    jwl1mn = (nlfc+1)*2;
                    nlft2 = nlf*2;
                    for jwl1 =( jwl1mn:2:nlft2)
                        jwl = jwl+2;
                        iwl(jwl-1) = iwl(jwl1-1);
                        iwl(jwl)   = iwl(jwl1);
                    end
                    nlf = jwl/2;
                else
                    break;
                end

            end

        end




    end

    for itt3 = (3:3:ntt3)
        ip1 = ipt(itt3-2);
        ip2 = ipt(itt3-1);
        ip3 = ipt(itt3);
        if(side(xd(ip1),yd(ip1),xd(ip2),yd(ip2),xd(ip3),yd(ip3))<0)
            ipt(itt3-2) = ip2;
            ipt(itt3-1) = ip1;
        end
    end
    nt = nt0;
    nl = nl0;
end

%Parámetros de Salida:

%nt:Número de triángulos.

%ipt: Índice de los vértices de los triángulos.

%nl: Número de lineas que conforman en borde de la triangular.

%ipl: Índice de los puntos que conforman las nl

%iwl,iwp,iwk: Vectores que guardan información de la malla.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                              %
%   Función para el intercambio de triángulos  %
%                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Parámetros de Entrada:

```

```
%x,y:Coordenadas (xi,yi).

%i1,i2,i3,i4: Índices de los puntos del cuadrilatero y la diagonal.

function[idx]=idxchg( x, y, i1, i2, i3, i4 )

  %Proceso preeliminar.

  x1 = x(i1);
  y1 = y(i1);
  x2 = x(i2);
  y2 = y(i2);
  x3 = x(i3);
  y3 = y(i3);
  x4 = x(i4);
  y4 = y(i4);

  %Cálculos.

  idx = 0;
  u3 = ( y2 - y3 ) * ( x1 - x3 ) - ( x2 - x3 ) * ( y1 - y3 );
  u4 = ( y1 - y4 ) * ( x2 - x4 ) - ( x1 - x4 ) * ( y2 - y4 );
  if ( 0 < u3 * u4 )

    u1 = (y3-y1)*(x4-x1)-(x3-x1)*(y4-y1);
    u2 = (y4-y2)*(x3-x2)-(x4-x2)*(y3-y2);

    a1sq = (x1-x3)^2+(y1-y3)^2;
    b3sq = a1sq;
    b1sq = (x4-x1)^2+(y4-y1)^2;
    a4sq = b1sq;
    c1sq = (x3-x4)^2+(y3-y4)^2;
    c2sq = c1sq;
    a2sq = (x2-x4)^2+(y2-y4)^2;
    b4sq = a2sq;
    b2sq = (x3-x2)^2+(y3-y2)^2;
    a3sq = b2sq;
    c3sq = (x2-x1)^2+(y2-y1)^2;
    c4sq = c3sq;

    s1sq = u1 * u1 / ( c1sq * max ( a1sq, b1sq ) );
    s2sq = u2 * u2 / ( c2sq * max ( a2sq, b2sq ) );
    s3sq = u3 * u3 / ( c3sq * max ( a3sq, b3sq ) );
    s4sq = u4 * u4 / ( c4sq * max ( a4sq, b4sq ) );

    if ( min ( s1sq, s2sq ) < min ( s3sq, s4sq ) )
      idx = 1;
    end

  end

end

%Parámetros de Salida:

%idx: Indicador del intercambio de triángulos.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                        %
%Encuentra el punto en el triángulo      %
%                                        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Parámetros de entrada:

%nt:Número de triángulos.

%ipt: Índice de los vértices de los triángulos.

%nl: Número de lineas que conforman en borde de la triangular.

%ipl: Índice de los puntos que conforman las nl.

%xii,yii: Punto (xi,yi) que se ubicará dentro del triágulo correspondiente.

%iwp,iwk: Vectores que guardan información de la malla.

function [iti,iwk, wk] =idlctn(xd, yd, nt, ipt, nl, ipl, xii, yii,iwk,wk)

    %Variables globales

    global idlc
    global nit

    global i1
    global i2
    global i3
    global idp
    global il1
    global il1t3
    global il2
    global ip1
    global ip2
    global ip3
    global isc
    global it0
    global it0t3
    global itipv
    global itsc
    global jiwk
    global jwk
    global ndp0
    global nl0
    global nt0
    global ntl
    global ntsci
    global x0
    global x1
    global x2
    global x3
    global xi
    global xmn
    global xmx
    global xs1
    global xs2
    global y0
    global y1
    global y2
    global y3
    global yi
    global ymn
```

```
65      global ymx
66      global ys1
67      global ys2
68      global ntsc
69      global idsc
70      ndp=length(xd);
71
72      %Vectores necesarios para los cálculos.
73
74      iwk=iwk(1:18*ndp,1);
75      wk=wk(1:8*ndp,1);
76
77      %Funciones para el cálculo.
78
79      function[lado]=side(u1,v1,u2,v2,u3,v3)
80          lado=(u1-u3)*(v2-v3) - (v1-v3)*(u2-u3);
81      end
82      function[resultado]=spdt(u1,v1,u2,v2,u3,v3)
83          resultado=(u1-u2)*(u3-u2) + (v1-v2)*(v3-v2);
84      end
85
86      %Proceso preeliminar.
87
88      ndp0 = ndp;
89      nt0 = nt;
90      nl0 = nl;
91      ntl = nt0 + nl0;
92      x0 = xii;
93      y0 = yii;
94
95      %Comenzamos con la ubicación dentro de la malla triangular.
96
97      if( nit==0)
98          nit = 1;
99          xmn = xd(1);
100         xmx = xmn;
101         ymn = yd(1);
102          ymx = ymn;
103
104         for idp = (2:ndp0)
105             xi = xd(idp);
106             yi = yd(idp);
107             xmn = min (xi,xmn);
108             xmx = max (xi,xmx);
109             ymn = min (yi,ymn);
110             ymx = max (yi,ymx);
111         end
112         xs1 = ( xmn + xmn + xmx ) / 3;
113         xs2 = ( xmn + xmx + xmx ) / 3;
114         ys1 = ( ymn + ymn + ymx ) / 3;
115         ys2 = ( ymn + ymx + ymx ) / 3;
116
117         ntsc(1:9) = 0;
118         idsc(1:9) = 0;
119         it0t3 = 0;
120         jwk = 0;
121
122         for it0 = (1:nt0)
123             it0t3 = it0t3 + 3;
124             i1 = ipt(it0t3-2);
125             i2 = ipt(it0t3-1);
126             i3 = ipt(it0t3);
127             xmn = min ([xd(i1),xd(i2),xd(i3)]);
128             xmx = max ([xd(i1),xd(i2),xd(i3)]);
```

```
            ymn = min ([yd(i1),yd(i2),yd(i3)]);
            ymx = max ([yd(i1),yd(i2),yd(i3)]);
            if (~(ymn>ys1))
                if(xmn<=xs1)
                    idsc(1) = 1;
                end
                if (xmx>=xs1 && xmn<=xs2)
                    idsc(2) = 1;
                end
                if (xmx>=xs2)
                    idsc(3) = 1;
                end

                if (~(ymx<ys1 || ymn>ys2))
                    if (xmn<=xs1)
                        idsc(4) = 1;
                    end
                    if (xmx>=xs1 && xmn<=xs2)
                        idsc(5) = 1;
                    end
                    if (xmx>=xs2)
                        idsc(6) = 1;
                    end

                     if (~(ymx<ys2))
                         if (xmn<=xs1)
                             idsc(7) = 1;
                         end
                         if (xmx>=xs1 && xmn<=xs2)
                             idsc(8) = 1;
                         end
                         if (xmx>=xs2)
                             idsc(9) = 1;
                         end

                     end
                else

                     if (~(ymx<ys2))
                         if (xmn<=xs1)
                             idsc(7) = 1;
                         end
                         if (xmx>=xs1 && xmn<=xs2)
                             idsc(8) = 1;
                         end
                         if (xmx>=xs2)
                             idsc(9) = 1;
                         end
                     end
                end
            else

                if (~(ymx<ys1 || ymn>ys2))
                    if (xmn<=xs1)
                        idsc(4) = 1;
                    end
                    if (xmx>=xs1 && xmn<=xs2)
                        idsc(5) = 1;
                    end
                    if (xmx>=xs2)
                        idsc(6) = 1;
                    end

                     if (~(ymx<ys2))
```

```
                        if (xmn<=xs1)
                            idsc(7) = 1;
                        end
                        if (xmx>=xs1 && xmn<=xs2)
                            idsc(8) = 1;
                        end
                        if (xmx>=xs2)
                            idsc(9) = 1;
                        end

                    end
                else

                    if (~(ymx<ys2))
                        if (xmn<=xs1)
                            idsc(7) = 1;
                        end
                        if (xmx>=xs1 && xmn<=xs2)
                            idsc(8) = 1;
                        end
                        if (xmx>=xs2)
                            idsc(9) = 1;
                        end
                    end
                end

            end

        %Guardamos le información de la malla trangular.

            for isc =(1:9)
                if ( idsc(isc) ~= 0 )
                    jiwk = 9*ntsc(isc) + isc;
                    iwk(jiwk) = it0;
                    ntsc(isc) = ntsc(isc) + 1;
                    idsc(isc) = 0;
                end
            end

            jwk = jwk + 4;
            wk(jwk-3) = xmn;
            wk(jwk-2) = xmx;
            wk(jwk-1) = ymn;
            wk(jwk) = ymx;
        end

        isc = 1;
        if (x0>=xs1)
            isc = isc + 1;
        end
        if (x0>=xs2)
            isc = isc + 1;
        end
        if (y0>=ys1)
            isc = isc + 3;
        end
        if (y0>=ys2)
            isc = isc + 3;
        end

        ntsci = ntsc(isc);
            if (~(ntsci<=0))
                jiwk = -9 + isc;
                for itsc = (1:ntsci)
```

```
                    jiwk = jiwk + 9;
                    it0 = iwk(jiwk);
                    jwk = it0*4;
                    if (x0<wk(jwk-3))
                        ok1=1;
                        continue;
                    end
                    if (x0>wk(jwk-2))
                        ok1=1;
                        continue;
                    end
                    if (y0<wk(jwk-1))
                        ok1=1;
                        continue;
                    end
                    if (y0>wk(jwk))
                        ok1=1;
                        continue;
                    end
                    it0t3 = it0*3;
                    ip1 = ipt(it0t3-2);
                    x1 = xd(ip1);
                    y1 = yd(ip1);
                    ip2 = ipt(it0t3-1);
                    x2 = xd(ip2);
                    y2 = yd(ip2);
                    if (side(x1,y1,x2,y2,x0,y0)<0)
                        ok1=1;
                        continue;
                    end
                    ip3 = ipt(it0t3);
                    x3 = xd(ip3);
                    y3 = yd(ip3);
                    if (side(x2,y2,x3,y3,x0,y0)<0 )
                        ok1=1;
                        continue;
                    end
                    if (side(x3,y3,x1,y1,x0,y0)<0 )
                        ok1=1;
                        continue;
                    end
                    ok1=2;
                    break;
                end
                if(ok1==2)
                    iti = it0;
                    itipv = it0;

                    % Obtenemos el índice del triángulo donde se encuentra
                    (xii,yii).

                    return;
                elseif(ok1==1)

                    for il1 = (1:nl0)
                        il1t3 = il1*3;
                        ip1 = ipl(il1t3-2);
                        x1 = xd(ip1);
                        y1 = yd(ip1);
                        ip2 = ipl(il1t3-1);
                        x2 = xd(ip2);
                        y2 = yd(ip2);
                        if (spdt(x2,y2,x1,y1,x0,y0)<0 )
                            ok2=2;
```

```
                        continue;
                    end
                    if (~(spdt(x1,y1,x2,y2,x0,y0)<0))
                        if (side(x1,y1,x2,y2,x0,y0)>0)
                            ok2=2;
                            continue;
                        end
                        il2 = il1;
                        ok2=1;
                        break;

                    else

                        il2 = mod(il1,nl0) + 1;
                        ip3 = ipl(3*il2-1);
                        x3 = xd(ip3);
                        y3 = yd(ip3);
                        if (spdt(x3,y3,x2,y2,x0,y0)<=0)
                            ok2=1;
                            break;
                        end
                    end

                    ok2=2;
                end
                if(ok2==2)
                    it0=1;
                    iti = it0;
                    itipv = it0;

                    % Obtenemos el índice del triángulo donde se
                    encuentra (xii,yii).

                    return;
                elseif(ok2==1)
                    it0 = il1*ntl + il2;
                    iti = it0;
                    itipv = it0;

                    % Obtenemos el índice del triángulo donde se
                    encuentra (xii,yii).

                    return;
                end


            end


        else

            for il1 = (1:nl0)
                il1t3 = il1*3;
                ip1 = ipl(il1t3-2);
                x1 = xd(ip1);
                y1 = yd(ip1);
                ip2 = ipl(il1t3-1);
                x2 = xd(ip2);
                y2 = yd(ip2);
                if (spdt(x2,y2,x1,y1,x0,y0)<0 )
                    ok2=2;
                    continue;
                end
                if (~(spdt(x1,y1,x2,y2,x0,y0)<0))
```

```
382                         if (side(x1,y1,x2,y2,x0,y0)>0)
383                             ok2=2;
384                             continue;
385                         end
386                         il2 = il1;
387                         ok2=1;
388                         break;
389 
390                     else
391 
392                         il2 = mod(il1,nl0) + 1;
393                         ip3 = ipl(3*il2-1);
394                         x3 = xd(ip3);
395                         y3 = yd(ip3);
396                         if (spdt(x3,y3,x2,y2,x0,y0)<=0)
397                             ok2=1;
398                             break;
399                         end
400                      end
401 
402                     ok2=2;
403                 end
404                 if(ok2==2)
405                     it0=1;
406                     iti = it0;
407                     itipv = it0;
408 
409                     % Obtenemos el índice del triángulo donde se encuentra
                        (xii,yii).
410 
411                     return;
412                 elseif(ok2==1)
413                     it0 = il1*ntl + il2;
414                     iti = it0;
415                     itipv = it0;
416 
417                     % Obtenemos el índice del triángulo donde se encuentra
                        (xii,yii).
418 
419                     return;
420                 end
421             end
422         else
423 
424             it0 = itipv;
425             for i=(1:1)
426                 if (~(it0>nt0))
427                     it0t3 = it0*3;
428                     ip1 = ipt(it0t3-2);
429                     x1 = xd(ip1);
430                     y1 = yd(ip1);
431                     ip2 = ipt(it0t3-1);
432                     x2 = xd(ip2);
433                     y2 = yd(ip2);
434                     if (side(x1,y1,x2,y2,x0,y0)<0 )
435                         ok3=1;
436                         break;
437                     end
438                     ip3 = ipt(it0t3);
439                     x3 = xd(ip3);
440                     y3 = yd(ip3);
441                     if (side(x2,y2,x3,y3,x0,y0)<0.0D+00 )
442                         ok3=1;
443                         break;
```

```
                    end
                    if (side(x3,y3,x1,y1,x0,y0)<0.0D+00 )
                        ok3=1;
                        break;
                    end
                    ok3=2;
                    break;
                else
                    il1 = floor(it0/ntl);
                    il2 = it0 - il1*ntl;
                    il1t3 = il1*3;
                    ip1 = ipl(il1t3-2);
                    x1 = xd(ip1);
                    y1 = yd(ip1);
                    ip2 = ipl(il1t3-1);
                    x2 = xd(ip2);
                    y2 = yd(ip2);
                    if (~(il2~=il1))
                        if (spdt(x1,y1,x2,y2,x0,y0)<0 )
                            ok3=1;
                            break;
                        end
                        if (spdt(x2,y2,x1,y1,x0,y0)<0.0D+00 )
                            ok3=1;
                            break;
                        end
                        if (side(x1,y1,x2,y2,x0,y0)>0.0D+00 )
                            ok3=1;
                            break;
                        end
                        ok3=2;
                        break;
                    else
                        if (spdt(x1,y1,x2,y2,x0,y0)>0 )
                            ok3=1;
                            break;
                        end
                        ip3 = ipl(3*il2-1);
                        x3 = xd(ip3);
                        y3 = yd(ip3);
                        if (spdt(x3,y3,x2,y2,x0,y0)<=0 )
                            ok3=2;
                            break;
                        end
                        ok3=1;
                        break;
                    end
                end
            end
            if(ok3==2)
                iti = it0;
                itipv = it0;

                % Obtenemos el índice del triángulo donde se encuentra
                (xii,yii).

                return;
            elseif(ok3==1)

                isc = 1;
                if (x0>=xs1)
                    isc = isc + 1;
                end
                if (x0>=xs2)
```

```
                    isc = isc + 1;
                end
                if (y0>=ys1)
                    isc = isc + 3;
                end
                if (y0>=ys2)
                    isc = isc + 3;
                end
                ntsci = ntsc(isc);
                if (~(ntsci<=0))
                    jiwk = -9 + isc;
                    for itsc = (1:ntsci)
                        jiwk = jiwk + 9;
                        it0 = iwk(jiwk);
                        jwk = it0*4;
                        if (x0<wk(jwk-3))
                            ok1=1;
                            continue;
                        end
                        if (x0>wk(jwk-2))
                            ok1=1;
                            continue;
                        end
                        if (y0<wk(jwk-1))
                            ok1=1;
                            continue;
                        end
                        if (y0>wk(jwk))
                            ok1=1;
                            continue;
                        end
                        it0t3 = it0*3;
                        ip1 = ipt(it0t3-2);
                        x1 = xd(ip1);
                        y1 = yd(ip1);
                        ip2 = ipt(it0t3-1);
                        x2 = xd(ip2);
                        y2 = yd(ip2);
                        if (side(x1,y1,x2,y2,x0,y0)<0)
                            ok1=1;
                            continue;
                        end
                        ip3 = ipt(it0t3);
                        x3 = xd(ip3);
                        y3 = yd(ip3);
                        if (side(x2,y2,x3,y3,x0,y0)<0 )
                            ok1=1;
                            continue;
                        end
                        if (side(x3,y3,x1,y1,x0,y0)<0 )
                            ok1=1;
                            continue;
                        end
                        ok1=2;
                        break;
                    end
                    if(ok1==2)
                        iti = it0;
                        itipv = it0;

                        % Obtenemos el índice del triángulo donde se
                        encuentra (xii,yii).

                        return;
```

```
570                     elseif(ok1==1)
571
572                         for il1 = (1:nl0)
573                             il1t3 = il1*3;
574                             ip1 = ipl(il1t3-2);
575                             x1 = xd(ip1);
576                             y1 = yd(ip1);
577                             ip2 = ipl(il1t3-1);
578                             x2 = xd(ip2);
579                             y2 = yd(ip2);
580                             if (spdt(x2,y2,x1,y1,x0,y0)<0 )
581                                 ok2=2;
582                                 continue;
583                             end
584                             if (~(spdt(x1,y1,x2,y2,x0,y0)<0))
585                                 if (side(x1,y1,x2,y2,x0,y0)>0)
586                                     ok2=2;
587                                     continue;
588                                 end
589                                 il2 = il1;
590                                 ok2=1;
591                              break;
592
593                             else
594
595                                 il2 = mod(il1,nl0) + 1;
596                                 ip3 = ipl(3*il2-1);
597                                 x3 = xd(ip3);
598                                 y3 = yd(ip3);
599                                 if (spdt(x3,y3,x2,y2,x0,y0)<=0)
600                                     ok2=1;
601                                     break;
602                                 end
603                             end
604
605                             ok2=2;
606                         end
607                         if(ok2==2)
608                             it0=1;
609                             iti = it0;
610                             itipv = it0;
611
612                             % Obtenemos el índice del triángulo donde se
                                encuentra (xii,yii).
613
614                             return;
615                         elseif(ok2==1)
616                             it0 = il1*ntl + il2;
617                             iti = it0;
618                             itipv = it0;
619
620                             % Obtenemos el índice del triángulo donde se
                                encuentra (xii,yii).
621
622                             return;
623                         end
624
625
626                     end
627
628
629                 else
630
631                     for il1 = (1:nl0)
```

```
632                         il1t3 = il1*3;
633                         ip1 = ipl(il1t3-2);
634                         x1 = xd(ip1);
635                         y1 = yd(ip1);
636                         ip2 = ipl(il1t3-1);
637                         x2 = xd(ip2);
638                         y2 = yd(ip2);
639                         if (spdt(x2,y2,x1,y1,x0,y0)<0 )
640                             ok2=2;
641                             continue;
642                         end
643                         if (~(spdt(x1,y1,x2,y2,x0,y0)<0))
644                             if (side(x1,y1,x2,y2,x0,y0)>0)
645                                 ok2=2;
646                                 continue;
647                             end
648                             il2 = il1;
649                             ok2=1;
650                             break;
651
652                         else
653
654                             il2 = mod(il1,nl0) + 1;
655                             ip3 = ipl(3*il2-1);
656                             x3 = xd(ip3);
657                             y3 = yd(ip3);
658                             if (spdt(x3,y3,x2,y2,x0,y0)<=0)
659                                 ok2=1;
660                                 break;
661                             end
662                         end
663
664                         ok2=2;
665                     end
666                     if(ok2==2)
667                         it0=1;
668                         iti = it0;
669                         itipv = it0;
670
671                         % Obtenemos el índice del triángulo donde se
                            encuentra (xii,yii).
672
673                         return;
674                     elseif(ok2==1)
675                         it0 = il1*ntl + il2;
676                         iti = it0;
677                         itipv = it0;
678                         % Obtenemos el índice del triángulo donde se
                            encuentra (xii,yii).
679
680                         return;
681                     end
682                 end
683
684             end
685
686     end
687 end
688
689 %Parámetros de salida:
690
691 %iti: índice del triángulo donde se encuentra el punto (xii,yii).
692
693 %iwk,wk: Vectores que guardan información de la triangulación.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                        %
%Interpolación Bivariada de Akima        %
%                                        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Parametros de entrada:

%xd,yd,zd: Vectores que contienes los los valores (xi,yi,zi) de cada punto.

%nt:Número de triángulos.

%ipt:Índice de los vértices de los triángulos.

%nl: Número de lineas que conforman en borde de la triangular.

%ipl:Índice de los puntos que conforman las nl.

%pdd: Vector que contiene la aproximación las derivadas parciales
zx,zy,zxx,zxy,zyy.

%iti:Índice del triángula donde se encuentra el punto que se va interpolar.

%xii,yii: Punto (xi,yi) que será interpoado.


function [zii] = idptip(xd,yd,zd,nt,ipt,nl,ipl,pdd,iti,xii,yii)

    %Variables globales.

    global idpi
    global itpv

    global   a
    global   aa
    global   ab
    global   ac
    global   act2
    global   ad
    global   adbc
    global   ap
    global   b
    global   bb
    global   bc
    global   bdt2
    global   bp
    global   c
    global   cc
    global   cd
    global   cp
    global   csuv
    global   d
    global   dd
    global   dlt
    global   dp
    global   dx
    global   dy
    global   g1
    global   g2
    global   h1
    global   h2
    global   h3
    global   i
    global   idp
```

```
64        global   il1
65        global   il2
66        global   it0
67        global   jipl
68        global   jipt
69        global   jpd
70        global   jpdd
71        global   kpd
72        global   lu
73        global   lv
74        global   ntl
75        global   p0
76        global   p00
77        global   p01
78        global   p02
79        global   p03
80        global   p04
81        global   p05
82        global   p1
83        global   p10
84        global   p11
85        global   p12
86        global   p13
87        global   p14
88        global   p2
89        global   p20
90        global   p21
91        global   p22
92        global   p23
93        global   p3
94        global   p30
95        global   p31
96        global   p32
97        global   p4
98        global   p40
99        global   p41
100       global   p5
101       global   p50
102       global   thsv
103       global   thus
104       global   thuv
105       global   thxu
106       global   u
107       global   v
108       global   x0
109       global   y0
110
111       %Vectores utilizados para el proceso.
112
113       pd=zeros(15,1);
114       x=zeros(3,1);
115       y=zeros(3,1);
116       z=zeros(3,1);
117       zii =0;
118       zu=zeros(3,1);
119       zuu=zeros(3,1);
120       zuv=zeros(3,1);
121       zv=zeros(3,1);
122       zvv=zeros(3,1);
123       %Proceso preeliminar.
124       it0 =floor( iti);
125       ntl = nt+nl;
126
127       %Comenzamos con la interpolación.
```

```
128
129       if(it0<=ntl)
130           if(it0~=itpv)
131               jipt = 3*(it0-1);
132               jpd = 0;
133               for i = (1:3)
134                   jipt = jipt+1;
135                   idp = ipt(jipt);
136                   x(i) = xd(idp);
137                   y(i) = yd(idp);
138                   z(i) = zd(idp);
139                   jpdd = 5*(idp-1);
140                   for kpd = (1:5)
141                       jpd = jpd+1;
142                       jpdd = jpdd+1;
143                       pd(jpd) = pdd(jpdd);
144                   end
145               end
146
147               %Realizamos la transformación UV.
148
149               x0 = x(1);
150               y0 = y(1);
151               a = x(2)-x0;
152               b = x(3)-x0;
153               c = y(2)-y0;
154               d = y(3)-y0;
155               ad = a*d;
156               bc = b*c;
157               dlt = ad-bc;
158               ap =  d/dlt;
159               bp = -b/dlt;
160               cp = -c/dlt;
161               dp =  a/dlt;
162
163               %Convertimos las derivadas parciales al sistema UV.
164
165               aa = a*a;
166               act2 = 2*a*c;
167               cc = c*c;
168               ab = a*b;
169               adbc = ad+bc;
170               cd = c*d;
171               bb = b*b;
172               bdt2 = 2*b*d;
173               dd = d*d;
174
175               for i = (1:3)
176                   jpd = 5*i;
177                   zu(i) = a*pd(jpd-4)+c*pd(jpd-3);
178                   zv(i) = b*pd(jpd-4)+d*pd(jpd-3);
179                   zuu(i) = aa*pd(jpd-2)+act2*pd(jpd-1)+cc*pd(jpd);
180                   zuv(i) = ab*pd(jpd-2)+adbc*pd(jpd-1)+cd*pd(jpd);
181                   zvv(i) = bb*pd(jpd-2)+bdt2*pd(jpd-1)+dd*pd(jpd);
182               end
183
184               %Calculamos los coeficientes del polinomio.
185
186               p00 = z(1);
187               p10 = zu(1);
188               p01 = zv(1);
189               p20 = 0.5*zuu(1);
190               p11 = zuv(1);
191               p02 = 0.5*zvv(1);
```

```
192             h1 = z(2)-p00-p10-p20;
193             h2 = zu(2)-p10-zuu(1);
194             h3 = zuu(2)-zuu(1);
195             p30 =  10*h1-4*h2+0.5*h3;
196             p40 = -15.0D+00*h1+7.0D+00*h2-h3;
197             p50 =   6*h1-3*h2+0.5*h3;
198             p5=p50;
199             h1 = z(3)-p00-p01-p02;
200             h2 = zv(3)-p01-zvv(1);
201             h3 = zvv(3)-zvv(1);
202             p03 =  10*h1-4*h2+0.5*h3;
203             p04 = -15*h1+7*h2-h3;
204             p05 =   6*h1-3*h2+0.5*h3;
205             lu = sqrt(aa+cc);
206             lv = sqrt(bb+dd);
207             thxu = atan2(c,a);
208             thuv = atan2(d,b)-thxu;
209             csuv = cos(thuv);
210             p41 = 5*lv*csuv/lu*p50;
211             p14 = 5*lu*csuv/lv*p05;
212             h1 = zv(2)-p01-p11-p41;
213             h2 = zuv(2)-p11-4*p41;
214             p21 =  3*h1-h2;
215             p31 = -2*h1+h2;
216             h1 = zu(3)-p10-p11-p14;
217             h2 = zuv(3)-p11-4.0D+00*p14;
218             p12 =  3*h1-h2;
219             p13 = -2*h1+h2;
220             thus = atan2(d-c,b-a)-thxu;
221             thsv = thuv-thus;
222             aa =  sin(thsv)/lu;
223             bb = -cos(thsv)/lu;
224             cc =  sin(thus)/lv;
225             dd =  cos(thus)/lv;
226             ac = aa*cc;
227             ad = aa*dd;
228             bc = bb*cc;
229             g1 = aa*ac*(3*bc+2*ad);
230             g2 = cc*ac*(3*ad+2*bc);
231             h1 = -aa*aa*aa*(5*aa*bb*p50+(4*bc+ad)*p41)-cc*cc*cc*(5*cc*dd*p05+(
                4*ad+bc)*p14);
232             h2 = 0.5*zvv(2)-p02-p12;
233             h3 = 0.5*zuu(3)-p20-p21;
234             p22 = (g1*h2+g2*h3-h1)/(g1+g2);
235             p32 = h2-p22;
236             p23 = h3-p22;
237             itpv = it0;
238
239             %Convertimos xii y yii al sistema UV.
240
241             dx = xii-x0;
242             dy = yii-y0;
243             u = ap*dx+bp*dy;
244             v = cp*dx+dp*dy;
245
246             %Evaluamos el polinomio.
247
248             p0 = p00+v*(p01+v*(p02+v*(p03+v*(p04+v*p05))));
249             p1 = p10+v*(p11+v*(p12+v*(p13+v*p14)));
250             p2 = p20+v*(p21+v*(p22+v*p23));
251             p3 = p30+v*(p31+v*p32);
252             p4 = p40+v*p41;
253
254             %Encontramos el punto buscado.
```

```
255
256             zii = p0+u*(p1+u*(p2+u*(p3+u*(p4+u*p5))));
257
258             return;
259
260         elseif(it0==itpv)
261
262             %Convertimos xii y yii al sistema UV.
263
264             dx = xii-x0;
265             dy = yii-y0;
266             u = ap*dx+bp*dy;
267             v = cp*dx+dp*dy;
268
269             %Evaluamos el polinomio.
270
271             p0 = p00+v*(p01+v*(p02+v*(p03+v*(p04+v*p05))));
272             p1 = p10+v*(p11+v*(p12+v*(p13+v*p14)));
273             p2 = p20+v*(p21+v*(p22+v*p23));
274             p3 = p30+v*(p31+v*p32);
275             p4 = p40+v*p41;
276
277             %Encontramos el punto buscado.
278
279             zii = p0+u*(p1+u*(p2+u*(p3+u*(p4+u*p5))));
280             return;
281         end
282
283     else
284         il1 = floor(it0/ntl);
285         il2 = floor(it0-il1*ntl);
286         if(il1==il2)
287             %Calculamos el valor de zii.
288             if(it0~=itpv)
289
290                 %Usamos las derivadas parciales.
291                 jipl = 3*(il1-1);
292                 jpd = 0;
293
294                 for i = (1:2)
295
296                     jipl = jipl+1;
297                     idp = ipl(jipl);
298                     x(i) = xd(idp);
299                     y(i) = yd(idp);
300                     z(i) = zd(idp);
301                     jpdd = 5*(idp-1);
302
303                     for kpd = (1:5)
304                         jpd = jpd+1;
305                         jpdd = jpdd+1;
306                         pd(jpd) = pdd(jpdd);
307                     end
308
309                 end
310
311                 %Creamos la conversión al sistema UV.
312
313                 x0 = x(1);
314                 y0 = y(1);
315                 a = y(2)-y(1);
316                 b = x(2)-x(1);
317                 c = -b;
318                 d = a;
```

```
                ad = a*d;
                bc = b*c;
                dlt = ad-bc;
                ap =  d/dlt;
                bp = -b/dlt;
                cp = -bp;
                dp =  ap;

                %Convertimos las derivadas parciales a sistema UV.

                aa = a*a;
                act2 = 2*a*c;
                cc = c*c;
                ab = a*b;
                adbc = ad+bc;
                cd = c*d;
                bb = b*b;
                bdt2 = 2*b*d;
                dd = d*d;

                for i = (1:2)
                    jpd = 5*i;
                    zu(i) = a*pd(jpd-4)+c*pd(jpd-3);
                    zv(i) = b*pd(jpd-4)+d*pd(jpd-3);
                    zuu(i) = aa*pd(jpd-2)+act2*pd(jpd-1)+cc*pd(jpd);
                    zuv(i) = ab*pd(jpd-2)+adbc*pd(jpd-1)+cd*pd(jpd);
                    zvv(i) = bb*pd(jpd-2)+bdt2*pd(jpd-1)+dd*pd(jpd);
                end

                %Calculamos los coeficientes del polinomio.

                p00 = z(1);
                p10 = zu(1);
                p01 = zv(1);
                p20 = 0.5D+00*zuu(1);
                p11 = zuv(1);
                p02 = 0.5D+00*zvv(1);
                h1 = z(2)-p00-p01-p02;
                h2 = zv(2)-p01-zvv(1);
                h3 = zvv(2)-zvv(1);
                p03 =  10*h1-4*h2+0.5*h3;
                p04 = -15*h1+7*h2-h3;
                p05 =   6*h1-3*h2+0.5*h3;
                h1 = zu(2)-p10-p11;
                h2 = zuv(2)-p11;
                p12 =  3*h1-h2;
                p13 = -2*h1+h2;
                p21 = 0;
                p23 = -zuu(2)+zuu(1);
                p22 = -1.5*p23;
                itpv = it0;

                %Covertimos xii y yii a sistema UV.

                dx = xii-x0;
                dy = yii-y0;
                u = ap*dx+bp*dy;
                v = cp*dx+dp*dy;

                %Evaluamos el polinomio.

                p0 = p00+v*(p01+v*(p02+v*(p03+v*(p04+v*p05))));
                p1 = p10+v*(p11+v*(p12+v*p13));
                p2 = p20+v*(p21+v*(p22+v*p23));
```

```
383
384                 %Encontramos el punto buscado.
385
386                 zii = p0+u*(p1+u*p2);
387                 return
388             elseif(it0==itpv)
389
390                 %Covertimos xii y yii a sistema UV.
391
392                 dx = xii-x0;
393                 dy = yii-y0;
394                 u = ap*dx+bp*dy;
395                 v = cp*dx+dp*dy;
396
397                 %Evaluamos el polinomio.
398
399                 p0 = p00+v*(p01+v*(p02+v*(p03+v*(p04+v*p05))));
400                 p1 = p10+v*(p11+v*(p12+v*p13));
401                 p2 = p20+v*(p21+v*(p22+v*p23));
402
403                 %Encontramos el punto buscado.
404
405                 zii = p0+u*(p1+u*p2);
406                 return
407             end
408         else
409             if ( it0 ~= itpv )
410
411                 %Usamos las derivadas parciales.
412
413                 jipl = 3*il2-2;
414                 idp = ipl(jipl);
415                 x(1) = xd(idp);
416                 y(1) = yd(idp);
417                 z(1) = zd(idp);
418                 jpdd = 5*(idp-1);
419
420                 for kpd = (1:5)
421                     jpdd = jpdd+1;
422                     pd(kpd) = pdd(jpdd);
423                 end
424
425                 %Calculamos los coeficientes del polinomio.
426
427                 p00 = z(1);
428                 p10 = pd(1);
429                 p01 = pd(2);
430                 p20 = 0.5*pd(3);
431                 p11 = pd(4);
432                 p02 = 0.5*pd(5);
433                 itpv = it0;
434
435                 %Convertimos xii y yii al sistema UV.
436                 u = xii-x(1);
437                 v = yii-y(1);
438
439                 %Evaluamos el polinomio.
440
441                  p0 = p00+v*(p01+v*p02);
442                  p1 = p10+v*p11;
443                  zii = p0+u*(p1+u*p20);
444             else
445
446                 %Convertimos xii y yii al sistema UV.
```

```
447                     u = xii-x(1);
448                     v = yii-y(1);
449
450                     %Evaluamos el polinomio.
451
452                      p0 = p00+v*(p01+v*p02);
453                      p1 = p10+v*p11;
454
455                      %Encontramos el punto buscado.
456
457                      zii = p0+u*(p1+u*p20);
458
459                 end
460
461             end
462         end
463     end
464
465     %Parámetros de Salida:
466
467     %zii:Valor de la Interpolación Bivariada de Akima en el punto (xii,yii).
468
469
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                        %
%Interpolación Bivariada de Akima        %
%                                        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Parámetros de entrada:

%ncp: Número de puntos cercanos para la estimación de las derivadas parciales.

%xd,yd,zd: Vectores que contiene los puntos (xi,yi,zi).

%xi,yi: Vectores que contienes los puntos para (xi,yi) para ser interpolados.

function [zi,iwk,wk] = idbvip(ncp,xd,yd,zd,xi,yi)

    %Variables globales.

    global idlc
    global idpi
    global nit
    global itpv

    %Variables para el cálculo.
    md=1;
    ndp=length(xd);
    nip=length(xi);
    zi=zeros(nip,1);
    iwk=zeros((max(31,27+ncp)*ndp+nip),1);

    %Proceso preeliminar.

    md0 = md;
    ncp0 = ncp;
    ndp0 = ndp;
    nip0 = nip;

    %Almacenamos en wk e iwk información necesaria para la interpolación.

    if ( md0 < 2 )

        iwk(1) = ncp0;
        iwk(2) = ndp0;

    end

    if ( md0 < 3 )

        iwk(3) = nip;

    end

    jwipt = 16;
    jwiwl = 6 * ndp0 + 1;
    jwiwk = jwiwl;
    jwipl = 24 * ndp0 + 1;
    jwiwp = 30 * ndp0 + 1;
    jwipc = 27 * ndp0 + 1;
    jwit0 = max ( 31, 27 + ncp0 ) * ndp0;

    %Obtenemos la malla triángular y su información.

    if( md0 == 1 )
        [nt, iwk11, nl, iwk12, iwk13, iwk14, wk]=idtang(xd,yd);
```

```
65          iwk(jwipt:jwipt+length(iwk11)-1)=iwk11;
66          iwk(jwipl:jwipl+length(iwk12)-1)=iwk12;
67          iwk(jwiwl:jwiwl+length(iwk13)-1)=iwk13;
68          iwk(jwiwp:jwiwp+length(iwk14)-1)=iwk14;
69          iwk(5) = nt;
70          iwk(6) = nl;
71          wk=[wk(1:ndp);zeros(7*ndp,1)];
72
73          if ( nt == 0 )
74              return;
75          end
76
77      end
78
79      %Obtenemos el índice de los puntos cercanos.
80
81      if ( md0 <= 1 )
82
83          [iwk21]=idcldp (xd, yd, ncp);
84          iwk(jwipc:jwipc+length(iwk21)-1)=iwk21;
85          if ( iwk(jwipc) == 0 )
86              return;
87          end
88
89      end
90
91      %Obtenemos la ubucación de cada (xi,yi) a ser interpolado en la malla
        triángular.
92
93      if ( md0 ~= 3 )
94
95          nit = 0;
96          jwit = jwit0;
97          for iip = (1:nip0)
98              jwit = jwit + 1;
99              [iwk(jwit),iwk41, wk]=idlctn( xd, yd, nt,iwk(jwipt:jwipt+3*nt-1),
                nl,iwk(jwipl:jwipl+3*nl-1), xi(iip), yi(iip),iwk(jwiwk:jwiwk+18*
                ndp-1),wk);
100             iwk(jwiwk:jwiwk+18*ndp-1)=iwk41;
101
102         end
103
104     %Obtenemos la aproximación a las derivadas parciales zx,zy,zxx,zxy,zyy.
105
106     end
107
108     [wk1]=idpdrv(xd,yd,zd,ncp0,iwk21);
109     wk=[wk1;wk(length(wk1)+1:8*ndp)];
110     itpv = 0;
111     jwit = jwit0;
112     iwk51=iwk(jwipt:jwipt+3*nt-1);
113     iwk52=iwk(jwipl:jwipl+3*nl-1);
114
115     %Obtenemos la interpolación de Bivariada de Akima de los los puntos
        (xi,yi).
116
117     for iip = (1:nip0)
118         jwit = jwit + 1;
119         [zi(iip) ]=idptip ( xd, yd, zd, nt, iwk51, nl, iwk52, wk1,iwk(jwit),
            xi(iip), yi(iip));
120     end
121
122 end
123
```

```
%Parámetros de Salida:

%zi: Vector que contiene el resultado de la Interpolación Bivariada de Akima.

%iwk,wk: Vectores que contiene información del proceso de interpolación.


```

Bibliografía

[1] Iwashita Yukinory.*Piecewise Polynomial Interplations.* OpenGamma Quantitative Research n.15, 2013.

[2] Burden, Richard L, J. Douglas Faires. *Análisis Numérico.* International Thomson Editores S.A. Séptima Edición, 2002.

[3] Hiroshi Akima. *A New Method of Interpolation and Smooth Curve Fitting Based on Local Procedures.* Journal of the Association for Computing Machinery,17(4):589–602,1970. 6.

[4] Hiroshi Akima, *A Method of Bivariate Interpolation and Smooth Surface Fitting for Irregulary Distributed Data Points.* ACM. Trans. Math. Software 4 1978.

[5] Escobar Freddy Humberto, Fundamentos de Ingeniería de Yacimientos.,2000.

[6] Curtis H. Whitson, Michel R. Brulé. *Phase Behavior.* Society of Petrolium Engineers, Monograph Volume 20 Henry L. Doherty Series, 2000.

[7] Pita Ruíz Claudio. *Cálculo Vectorial.* Prentice Hall Hispanoamericana S.A. Primera Edición, 1995.

[8] Peng, D.-Y.; Robinson, D.B. (1976) "A new two-constant equation of state" Ind. Eng. Chem. Fundamentals, 15, pp. 59-64.