TAREA 10 - METODOS NUMERICOS

Guillermo Segura Gómez
Centro de Investigación en Matemáticas
Métodos Numéricos
29 de octubre de 2023

1 Introducción

La interpolación es una herramienta muy útil dentro del marco de la matemática aplicada. Nos permite poder conocer y aproximar funciones a un conjunto de datos conocidos. En la tarea pasada revisamos la introducción a este tema, revisando los polinomios de Taylor y lagrange, además de los métodos de interpolación de Neville y diferencias divididas. Estos métodos son poderosos en el sentido que nos permiten inferir datos planteando un modelo de interpolación simple. Sin embargo, estos métodos tienen fallas. Por ejemplo, en muchos sistemas físicos a menudo se quiere que la función aproximada sea una función continua y diferenciable, lo que normalmente se conoce como función "suave" [1]. Es en este contexto que vamos a revisar técnicas de interpolación mas avanzadas, las cuales nos van a permitir considerar funciones interpoladas suaves, teniendo en cuenta sus derivadas a la hora de hacer los cálculos.

2 Problema 1

Consideremos la función $\sin(x)$: a) Genera un código basado en la interpolación de Hermite con diferencias divididas visto en clase (libro Burden). b) Con los valores de la siguiente tabla estima el valor de $\sin(0.34)$ utilizando interpolación de Hermite con diferencias divididas. Halla el error absoluto de la aproximacimación con el valor correcto. c) A la tabla añade

x	$\sin x$	$D_x \sin x = \cos x$
0.30	0.29552	0.95534
0.32	0.31457	0.94924
0.35	0.34290	0.93937

$\sin(0.33) = 0.32404$ y $\cos(0.33) = 0.94604$ y recalcula el valor de $\sin(0.34)$. Halla el error absoluto, ¿Qué puedes concluir?

2.1 Pseudocódigo

El pseudocódigo del programa de diferencias divididas de hermite se muestra a continuación.

```
Function calculaDiferenciasDivididasHermite(n: Integer, x: Array of Double, z
    : Array of Double, f: Array of Double, fp: Array of Double, F: 2D Array of
     Double)
    zlen <- 2 * n

    // Duplicar los puntos x y f(x)
    For i from 0 to n-1 with step of 1 and j from 0 to 2*(n-1) with step of 2
     Do
        z[j] <- x[i]
        z[j+1] <- x[i]

        F[j][0] <- f[i]
        F[j+1][0] <- f[i]
    EndFor

    // Llenar la fila de las derivadas (2da columna de F)
    For i from 1 to zlen-1 Do
        If i mod 2 = 0 Then
```

```
            F[i][1] <- (F[i][0] - F[i-1][0]) / (z[i] - z[i-1])
        Else
            F[i][1] <- fp[i/2]
        EndIf
    EndFor

    // Llenar las demás columnas de F
    For j from 2 to zlen-1 Do
        For i from j to zlen-1 Do
            F[i][j] <- (F[i][j-1] - F[i-1][j-1]) / (z[i] - z[i-j])
        EndFor
    EndFor

    // Imprimir la tabla
    For i from 0 to zlen-1 Do
        For j from 0 to i Do
            print F[i][j] with 5 decimal places followed by a space
        EndFor
        print newline
    EndFor
EndFunction
```

Listing 1: Algoritmo de Diferencias Divididas de Hermite

2.2 Inciso a

El código realizado para realizar la interpolación de Hermite se encuentra en el archivo **Hermite.c**. Para probar su funcionamiento, lo probaremos con un ejemplo del libro Burden página 104 [1].

```
guillermo_sego@192 Tarea10 % gcc Hermite.c -o Hermite
guillermo_sego@192 Tarea10 % ./Hermite
Ingrese el número de puntos n: 3
Ingrese los valores de x, f(x) y f'(x):
x[0] = 1.3
f[0] = 0.6200860
f'[0] = -0.5220232
x[1] = 1.6
f[1] = 0.4554022
f'[1] = -0.5698959
x[2] = 1.9
f[2] = 0.2818186
f'[2] = -0.5811571
Ingrese el valor de x para interpolar: 1.5
0.62009
0.62009 -0.52202
0.45540 -0.54895 -0.08974
0.45540 -0.56990 -0.06983 0.06637
0.28182 -0.57861 -0.02905 0.06797 0.00267
0.28182 -0.58116 -0.00848 0.06857 0.00100 -0.00277
El valor interpolado en x = 1.5000000 es: 0.5118277
```

Listing 2: Ejecución inciso a

Se puede observar que se imprime la tabla de diferencias divididas con los valores de los coeficientes de Hermite. Luego se calcula el valor interpolado para x. El valor de x es el mismo proporcionado por el ejemplo en el libro burden

2.3 Inciso b

Ahora calculamos los valores proporcionados en la tabla para la función $\sin(x)$ con x = 0.34.

```
guillermo_sego@192 Tarea10 % gcc Hermite.c -o Hermite
guillermo_sego@192 Tarea10 % ./Hermite
Ingrese el número de puntos n: 3
Ingrese los valores de x, f(x) y f'(x):
x[0] = 0.30
f[0] = 0.29552
f'[0] = 0.95534
x[1] = 0.32
f[1] = 0.31457
f'[1] = 0.94924
x[2] = 0.35
f[2] = 0.34290
f'[2] = 0.93937
Ingrese el valor de x para interpolar: 0.34
0.29552
0.29552 0.95534
0.31457 0.95250 -0.14200
0.31457 0.94924 -0.16300 -1.05000
0.34290 0.94433 -0.16356 -0.01111 20.77778
0.34290 0.93937 -0.16544 -0.06296 -1.03704 -436.29630
El valor interpolado en x = 0.3400000 es: 0.3334889
El error absoluto de la aproximación de sin(0.34) es: 0.000002
```

Listing 3: Ejecución inciso b

2.4 Inciso c

Vamos a añadir $\sin(0.33) = 0.32404$ y $\cos(0.33) = 0.94604$

```
guillermo_sego@192 Tarea10 % gcc Hermite.c -o Hermite
guillermo_sego@192 Tarea10 % ./Hermite
Ingrese el número de puntos n: 4
Ingrese los valores de x, f(x) y f'(x):
x[0] = 0.30
f[0] = 0.29552
f'[0] = 0.95534
x[1] = 0.32
f[1] = 0.31457
f'[1] = 0.94924
x[2] = 0.35
f[2] = 0.34290
f'[2] = 0.93937
x[3] = 0.33
f[3] = 0.32404
f'[3] = 0.94604
Ingrese el valor de x para interpolar: 0.34
0.29552
0.29552 0.95534
0.31457 0.95250 -0.14200
0.31457 0.94924 -0.16300 -1.05000
0.34290 0.94433 -0.16356 -0.01111 20.77778
0.34290 0.93937 -0.16544 -0.06296 -1.03704 -436.29630
0.32404 0.94300 -0.18150 -1.60556 -154.25926 -5107.40741 -155703.70370
```

```
0.32404 0.94604 -0.15200 -1.47500 13.05556 16731.48148 727962.96297
    29455555.55569
El valor interpolado en x = 0.3400000 es: 0.3334978
El error absoluto de la aproximación de sin(0.34) es: 0.000011%
```

Listing 4: Ejecución inciso c

Añadiendo un punto extra a la interpolación de Hermite el error absoluto del cálculo aumento, ya que con tres puntos tenemos 0.000002 y con cuatro 0.000011. Me parece una observación bastante interesante, ya que aunque intuitivamente se pensaría que el error disminuiría al añadir mas puntos, en este caso fue el contrario. De esta manera se concluye que no siempre tener mas puntos dentro de una interpolación puede aumentar la precisión del programa.

3 Problema 2

Genera un código en C que realice interpolación cúbica natural, puedes basarte en el algoritmo visto en clase. Con este algoritmo recrea la parte superior de la imagen del pato dado por los puntos de la tabla.

El pseudocódigo del programa de spin cúbico natural

```
PROCEDURE Spin3Natural(n: INTEGER; x, f, a, b, c, d: ARRAY OF REAL)
VAR
    alpha, h, l, mu, z: ARRAY OF REAL;
    i, j: INTEGER;
BEGIN

    (* Paso 1 *)
    FOR i := 0 TO n-1 DO
        h[i] := x[i+1] - x[i];
    END FOR;

    (* Paso 2 *)
    FOR i := 1 TO n-1 DO
        alpha[i] := (3.0/h[i]) * (f[i+1] - f[i]) - (3.0/h[i-1]) * (f[i] - f[i
    -1]);
    END FOR;

    (* Paso 3 *)
    l[0] := 1.0;
    mu[0] := 0.0;
    z[0] := 0.0;

    (* Paso 4 *)
    FOR i := 1 TO n-1 DO
        l[i] := 2.0 * (x[i+1] - x[i-1]) - h[i-1] * mu[i-1];
        mu[i] := h[i] / l[i];
        z[i] := (alpha[i] - h[i-1] * z[i-1]) / l[i];
    END FOR;

    (* Paso 5 *)
    l[n] := 1.0;
    z[n] := 0.0;
    c[n] := 0.0;
```

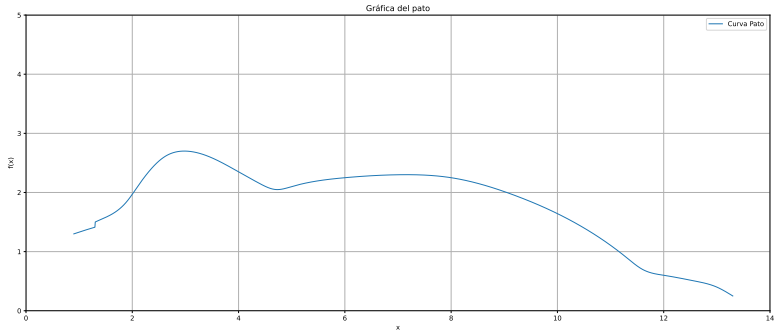

Figure 1: Curva generada por la interpolación cúbica natural.

```
    (* Paso 6 *)
    FOR j := n-1 DOWN TO 0 DO
        c[j] := z[j] - mu[j] * c[j+1];
        b[j] := (f[j+1] - f[j]) / h[j] - h[j] * (c[j+1] + 2.0 * c[j]) / 3.0;
        d[j] := (c[j+1] - c[j]) / (3.0 * h[j]);
        a[j] := f[j];
    END FOR;

END PROCEDURE;
```

Listing 5: Spline Cúbico Natural Pseudocódigo

El código del programa esta en el archivo **Spin3natural.c**. En el código se tienen comentados algunas lineas de pruebas en las que se probó el código utilizando los ejemplos vistos en el libro burden. Para el cálculo del polinomio se carga la función $f(x)$ en conjunto con x. Para la gráfica se utiliza la función *listing* la cual genera n números en un intervalo. Guardamos los resultados en un archivo txt y después los graficamos en python.

La ejecución del programa se muestra a continuación

```
guillermo_sego@192 Tarea10 % gcc Spin3natural.c -o Spin3natural
guillermo_sego@192 Tarea10 % ./Spin3natural
```

Listing 6: Ejecución del método del polinomio de lagrange

A continuación se muestra la gráfica del pato.

4 Problema 3

Genera un código en C que realice interpolación cúbica fijo o sujeto, puedes basarte en el algoritmo visto en clase. Con este algoritmo recrea la parte superior de la imagen del perro dado por los puntos de la tabla (puntos y derivadas de 3 secciones):

El pseudocódigo del programa de spin cúbico condicionado (fijo)

```
PROCEDURE Spin3Condicionado(n: INTEGER, x, f: ARRAY OF REAL, FP0, FPN: REAL,
    a, b, c, d: ARRAY OF REAL)
VAR
    alpha, h, l, mu, z: ARRAY OF REAL;
    i, j: INTEGER;
```

```
BEGIN

    (* Paso 1 *)
    FOR i := 0 TO n-1 DO
        h[i] := x[i+1] - x[i];
    END FOR;

    (* Paso 2 *)
    alpha[0] := 3.0*(f[1]-f[0])/h[0] - 3.0*FP0;
    alpha[n] := 3.0*FPN - 3.0*(f[n]-f[n-1])/h[n-1];

    (* Paso 3 *)
    FOR i := 1 TO n-1 DO
        alpha[i] := (3.0/h[i]) * (f[i+1] - f[i]) - (3.0/h[i-1]) * (f[i] - f[i
    -1]);
    END FOR;

    (* Paso 4 *)
    l[0] := 2.0 * h[0];
    mu[0] := 0.5;
    z[0] := alpha[0] / l[0];

    (* Paso 5 *)
    FOR i := 1 TO n-1 DO
        l[i] := 2.0 * (x[i+1] - x[i-1]) - h[i-1] * mu[i-1];
        mu[i] := h[i] / l[i];
        z[i] := (alpha[i] - h[i-1] * z[i-1]) / l[i];
    END FOR;

    (* Paso 6 *)
    l[n] := h[n-1] * (2.0 - mu[n-1]);
    z[n] := (alpha[n] - h[n-1] * z[n-1]) / l[n];
    c[n] := z[n];

    (* Paso 7 *)
    FOR j := n-1 DOWN TO 0 DO
        c[j] := z[j] - mu[j] * c[j+1];
        b[j] := (f[j+1] - f[j]) / h[j] - h[j] * (c[j+1] + 2.0 * c[j]) / 3.0;
        d[j] := (c[j+1] - c[j]) / (3.0 * h[j]);
        a[j] := f[j];
    END FOR;

END PROCEDURE;
```

Listing 7: Spline Cúbico Condicionado Pseudocódigo

El código del programa esta en el archivo **Spin3condicionado.c**. Al igual que el problema anterior, el código se tienen comentados algunas lineas de pruebas en las que se probó el código utilizando los ejemplos vistos en el libro burden. El funcionamiento es prácticamente el mismo que el problema pasado. Se calcularon las tres curvas por separado. Posteriormente se graficaron en pyhton.

La ejecución del programa se muestra a continuación

```
guillermo_sego@192 Tarea10 % gcc Spin3condicionado.c -o Spin3condicionado
guillermo_sego@192 Tarea10 % ./Spin3condicionado
```

Listing 8: Ejecución del método del polinomio de lagrange

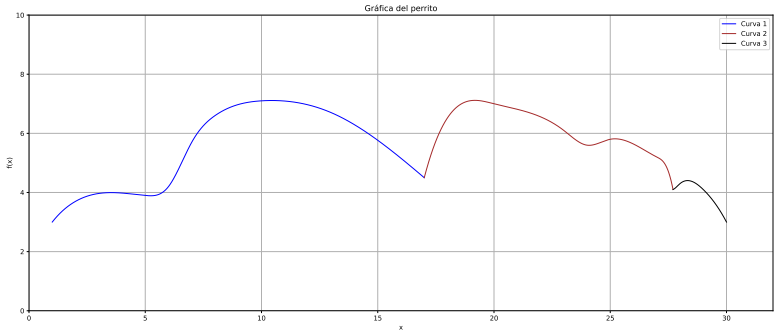

Figure 2: Curva generada por la interpolación cúbica condicionada.

A continuación se muestra la gráfica del perro.

5 Conclusión

Después de explorar técnicas de interpolación mas avanzadas, el método de Hermite y las interpolaciones cúbicas, es evidente la riqueza y versatilidad que ofrecen estas herramientas. Cada método, con sus características distintivas, brinda soluciones adaptadas a diferentes escenarios y necesidades. Por ejemplo, La interpolación cúbica, tanto en su variante natural como condicionada, se destaca por su suavidad y adaptabilidad, siendo los únicos datos que necesitamos los valores de x y la función en los puntos a evaluar. Por otro lado, el método de Hermite proporciona un enfoque más detallado, teniendo en cuenta no solo los valores sino también las derivadas, permitiendo tener funciones "suaves" tal y como se mencionó en la introducción. Al igual que con otras técnicas de interpolación, es vital ser críticos y conscientes de las limitaciones. Aunque estos métodos ofrecen estimaciones poderosas y detalladas, la esencia de la información original no siempre puede ser capturada en su totalidad. Además del ejercicio 1 inciso c, concluimos que no siempre una mayor cantidad de puntos representa una mejora en la precisión de un sistema.

References

[1] R. L. Burden, J. D. Faires, and A. M. Burden, *Numerical analysis*. Cengage learning, 2015.