Tarea 10 - Métodos de interpolación
Métodos Numéricos

Rafael Alejandro García Ramírez

29 de octubre de 2023

1. Introducción

Como ya se ha visto anteriormente, al momento de necesitar un la información al respecto de un sistema que se sabe tiene o debe tener un comportamiento específico se suele utilizar un método de interpolación para buscar el dato más ajustado al sistema, sin embargo esto supone a priori establecer o fijar el comportamiento teórico o cuasi real del sistema por lo que se necesita tener conocimiento previo de su naturaleza.

Cuando se desconoce esto, una aproximación al problema es considerar características de los datos que necesitamos preservar; quizás no sepamos bien qué comportamiento tienen o aparentan tener los datos, pero sí sabemos que necesitamos que, por ejemplo, su derivada sea la misma en cada punto conservando características (como la velocidad de un objeto) que de otra forma se perderían realizando otros métodos de interpolación.

Con esto en mente, vamos a ver dos métodos en particular, los cuales son la interpolación de Hermite y los splines cúbicos. El primero justamente preserva la característica de que las derivadas son la misma en todos los puntos, lo cuál brinda un mayor control local de los datos (esto es, que entre más cercano sea la interpolación a alguno de los datos más buena es la aproximación, es decir información del vecindario que nos indica la pendiente o dirección de los datos próximos) y los splines que se centran más en el control local y se dividen en dos: los naturales que solo toma en cuenta los datos dados y los fijos o condicionados que reciben un conjunto de derivadas en los puntos sobre los cuales se desea mantener fijo sus pendientes localmente.

2. Pseudocódigos

2.1. Interpolador de Hermite usando diferencias divididas

Algorithm 1 Interpolación de Hermite usando diferencias divididas

Require: $N, x, y, derivadas, punto$
Ensure: Valor interpolado P

```
1: function HERMITE(N, x, y, derivadas, punto)
2:
3:     Q ← generar matriz de ceros N × N                ▷ Reservar memoria para la matriz Q
4:
5:     for i ← 0 hasta N − 1 do                         ▷ Llenamos la matriz con los valores de y y las derivadas
6:         Q[i][0] ← y[i]
7:         Q[i][1] ← derivadas[i]
8:         if i ≠ 0 then
9:             Q[i][1] ← (Q[i][0] − Q[i − 1][0])/(x[i] − x[i − 1])
10:        end if
11:    end for
12:
13:    for i ← 2 hasta N − 1 do                         ▷ Calculamos los coeficientes
14:        for j ← 2 hasta i do
15:            Q[i][j] ← (Q[i][j − 1] − Q[i − 1][j − 1])/(x[i] − x[i − j])
16:        end for
17:    end for
18:
19:    producto ← 1.0, P ← 0.0                          ▷ Calculamos el polinomio
20:    for i ← 0 hasta N − 1 do
21:        producto ← Q[i][i]
22:        for j ← 0 hasta i do
23:            producto ← producto · (punto − x[j])
24:        end for
25:        P ← P + producto
26:    end for
27:    return P
28: end function
```

2.2. Spline Natural

Algorithm 2 Interpolación Spline Natural

Require: $N, x, y, punto$
Ensure: Valor interpolado *spline*

function SplineNatural($N, x, y, punto$)

    $a, b, c, d, h, \alpha \leftarrow$ inicializar arreglos de tamaño $N-1$ ▷ Calcula las diferencias de x y α
    for $i \leftarrow 0$ **hasta** $N-2$ **do**
        $h[i] \leftarrow x[i+1] - x[i]$
        $alpha[i] \leftarrow (3.0/h[i]) \cdot (y[i+1] - y[i])$
        if $i == 0$ **then**
            $alpha[i] \leftarrow \alpha[i] - (3.0/h[i]) \cdot (y[i] - y[i-1])$
        else
            $alpha[i] \leftarrow \alpha[i] - (3.0/h[i-1]) \cdot (y[i] - y[i-1])$
        end if
    end for

    $l, u, w \leftarrow$ inicializar arreglos de tamaño N
    $l[0], u[0], w[0] \leftarrow 1.0, 0.0, 0.0$

    for $i \leftarrow 1$ **hasta** $N-2$ **do** ▷ Calcula coeficientes l, u, y w
        $l[i] \leftarrow 2.0 \cdot (x[i+1] - x[i-1]) - h[i-1] \cdot u[i-1]$
        $u[i] \leftarrow h[i]/l[i]$
        $w[i] \leftarrow (alpha[i] - h[i-1] \cdot w[i-1])/l[i]$
    end for
    $c[N-1] \leftarrow 0.0$
    $l[N-1] \leftarrow 1.0$

    for $i \leftarrow N-2$ **hasta** 0 **en orden inverso do** ▷ Calcula coeficientes c y retrocede para encontrar a, b, y d
        $c[i] \leftarrow w[i] - u[i] \cdot c[i+1]$
        $b[i] \leftarrow (y[i+1] - y[i])/h[i] - h[i] \cdot (c[i+1] + 2.0 \cdot c[i])/3.0$
        $d[i] \leftarrow (c[i+1] - c[i])/(3.0 \cdot h[i])$
        $a[i] \leftarrow y[i]$
    end for

    $intervalo \leftarrow 0$ ▷ Encuentra el intervalo
    for $i \leftarrow 0$ **hasta** $N-2$ **do**
        if $x[i] \leq punto \leq x[i+1]$ **then**
            $intervalo \leftarrow i$
            break
        end if
    end for

    $t \leftarrow punto - x[intervalo]$
    $spline \leftarrow a[intervalo] + b[intervalo] \cdot t + c[intervalo] \cdot t \cdot t + d[intervalo] \cdot t \cdot t \cdot t$

    return *spline*
end function

2.3. Spline Condicionado

Algorithm 3 Interpolación Spline Condicionado

Require: $N, x, y, punto$
Ensure: Valor interpolado $resultado$

function SPLINE_CONDICIONADO($N, x, y, punto$)

    $a, b, c, d, h, \alpha \leftarrow$ inicializar arreglos de tamaño $N-1$ ▷ Calcula las diferencias de x y α
    for $i \leftarrow 0$ **hasta** $N-2$ **do**
        $h[i] \leftarrow x[i+1] - x[i]$
        $alpha[i] \leftarrow (3.0/h[i]) \cdot (y[i+1] - y[i])$
        if $i == 0$ **then**
            $alpha[i] \leftarrow \alpha[i] - (3.0/h[i]) \cdot (y[i] - y[i-1])$
        else
            $alpha[i] \leftarrow \alpha[i] - (3.0/h[i-1]) \cdot (y[i] - y[i-1])$
        end if
    end for

    $l, u, w \leftarrow$ inicializar arreglos de tamaño N
    $l[0], u[0], w[0] \leftarrow 1.0, 0.0, 0.0$

    for $i \leftarrow 1$ **hasta** $N-2$ **do** ▷ Calcula coeficientes l, u, y w
        $l[i] \leftarrow 2 \cdot (h[i-1] + h[i]) - h[i-1] \cdot u[i-1]$
        $u[i] \leftarrow h[i]/l[i]$
        $w[i] \leftarrow (alpha[i] - alpha[i-1]) - h[i-1] \cdot w[i-1]$
        $w[i] \leftarrow w[i]/l[i]$
    end for
    $l[N-1] \leftarrow 1.0$
    $u[N-1] \leftarrow 0.0$
    $w[N-1] \leftarrow 0.0$

    for $i \leftarrow N-2$ **hasta** 0 **en orden inverso do** ▷ Calcula coeficientes c, b, d y a
        $c[i] \leftarrow w[i] - u[i] \cdot c[i+1]$
        $b[i] \leftarrow (y[i+1] - y[i])/h[i] - h[i] \cdot (c[i+1] + 2 \cdot c[i])/3.0$
        $d[i] \leftarrow (c[i+1] - c[i])/(3.0 \cdot h[i])$
        $a[i] \leftarrow y[i]$
    end for

    $intervalo \leftarrow 0$ ▷ Encuentra el intervalo correspondiente al punto
    for $i \leftarrow 0$ **hasta** $N-2$ **do**
        if $x[i] \leq punto \leq x[i+1]$ **then**
            $intervalo \leftarrow i$
            romper el ciclo
        end if
    end for

    $dx \leftarrow punto - x[intervalo]$
    $resultado \leftarrow a[intervalo] + b[intervalo] \cdot dx + c[intervalo] \cdot dx^2 + d[intervalo] \cdot dx^3$
    return $resultado$
end function

3. Resultados

A continuación presentamos la resolución de los problemas presentados en la tarea:

1. a. El código para la interpolación de Hermite usando diferencias divididas es el siguiente

```
double hermite(int N, double *x, double *y, double *derivadas, double punto) {
    // abrimos espacio en la memoria
    double **Q = generar_matriz(N);

    // inicializamos la matriz en ceros
    for(int i = 0; i < N; i++) for(int j = 0; j < N; j++) Q[i][j] = 0.0;

```

```
    // llenamos la matriz con los valores de y y las derivadas
    for(int i = 0; i < N; i++){
        Q[i][0] = y[i];
        Q[i][1] = derivadas[i];
        if(i != 0) Q[i][1] = (Q[i][0] - Q[i - 1][0]) / (x[i] - x[i - 1]);
    }

    // calculamos los coeficientes
    for(int i = 2; i < N; i++)
         for (int j = 2; j <= i; j++)
             Q[i][j] = (Q[i][j - 1] - Q[i - 1][j - 1]) / (x[i] - x[i - j]);

    // calculamos el polinomio
    double producto, P = 0.0;
    for(int i = 0; i < N; i++){
        producto = Q[i][i];
        for(int j = 0; j < i; j++) producto *= (punto - x[j]);
        P += producto;
    }

    // liberamos la memoria
    liberar_matriz(N, Q);

    return P;
}
```

b. Utilizando los valores $x = \{0.30, 0.32, 0.35\}$, $y = \{0.29552, 0.31457, 0.34290\}$ para $\sin x_j$ y $D_x \sin(x_j) = \{0.95534, 0.94924, 0.93937\}$ se obtiene el valor $\sin 0.34 \approx 0.333489$ con un valor real de $\sin 0.34 \cong 0.333487$ tenemos así un error absoluto de $\text{Error}_{abs} = 2.24119 \times 10^{-6}$.

c. Utilizando los valores extras que se nos proporcionan obtenemos que el valor interpolado es $\sin 0.34 \approx 0.333481$ con un error $\text{Error}_{abs} = 5.7588 \times 10^{-6}$ que es mucho mayor que en el caso donde no se tenía este punto adicional, esto es debido principalmente a naturaleza de los datos y que se acotan los valores a 5 decimales. Concluimos entonces que no por tener una mayor cantidad de datos, esto significa obtener una mayor precisión.

2. El código utilizado para generar el spline natural es el siguiente

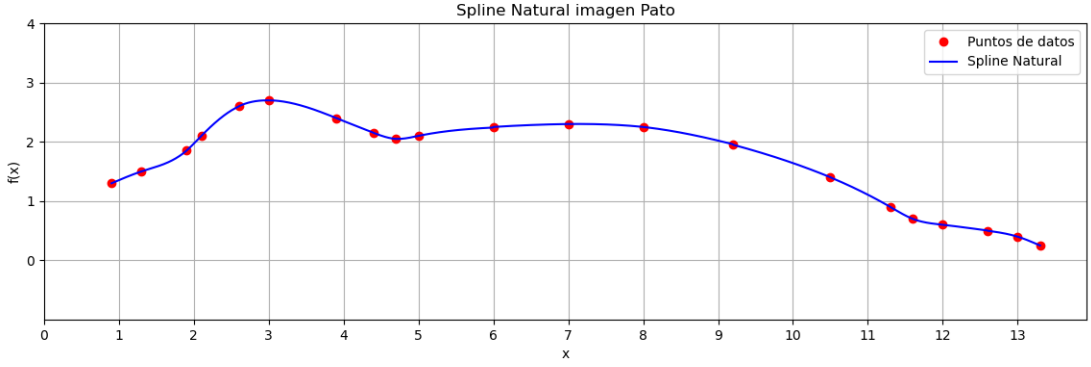
```
double spline_natural(int N, double *x, double *y, double punto) {
    double *a = (double *)malloc(N * sizeof(double));
    double *b = (double *)malloc(N * sizeof(double));
    double *c = (double *)malloc(N * sizeof(double));
    double *d = (double *)malloc(N * sizeof(double));

    double *h = (double *)malloc((N - 1) * sizeof(double));
    double *alpha = (double *)malloc((N - 1) * sizeof(double));

    // Calcular diferencias de x y alpha
    for (int i = 0; i < N - 1; i++) {
        h[i] = x[i + 1] - x[i];
        alpha[i] = (3.0 / h[i]) * (y[i + 1] - y[i]) - (3.0 / (i == 0 ? h[i] : h[i - 1])) * (y[i] -
     y[i - 1]);
    }

    double *l = (double *)malloc(N * sizeof(double));
    double *u = (double *)malloc(N * sizeof(double));
    double *w = (double *)malloc(N * sizeof(double));

    l[0] = 1.0;
    u[0] = 0.0;
    w[0] = 0.0;

    // Calcular coeficientes l, u y w
    for (int i = 1; i < N - 1; i++) {
        l[i] = 2.0 * (x[i + 1] - x[i - 1]) - h[i - 1] * u[i - 1];
        u[i] = h[i] / l[i];
        w[i] = (alpha[i] - h[i - 1] * w[i - 1]) / l[i];
    }

    c[N - 1] = 0.0;
    l[N - 1] = 1.0;

    // Calcular coeficiente c y retroceder para encontrar a, b y d
    for (int i = N - 2; i >= 0; i--) {
        c[i] = w[i] - u[i] * c[i + 1];
        b[i] = (y[i + 1] - y[i]) / h[i] - h[i] * (c[i + 1] + 2.0 * c[i]) / 3.0;
        d[i] = (c[i + 1] - c[i]) / (3.0 * h[i]);
        a[i] = y[i];
    }

```

```
    int intervalo = 0;
    for (int i = 0; i < N - 1; i++) {
        if (x[i] <= punto && punto <= x[i + 1]) {
            intervalo = i;
            break;
        }
    }

    double t = punto - x[intervalo];
    double spline = a[intervalo] + b[intervalo] * t + c[intervalo] * t * t + d[intervalo] * t * t
    * t;

    free(a);
    free(b);
    free(c);
    free(d);
    free(h);
    free(alpha);
    free(l);
    free(u);
    free(w);

    return spline;
}
```

Cabe resaltar, que para el caso donde se tenga interés en construir los splines por secciones uno puede cambiar los argumentos de la función para que reciban arrays donde se guardan los coeficientes a_j, b_j, c_j y d_j. Esto fue justamente lo que se hizo para obtener los valores de los coeficientes en la siguiente tabla

j	x_j	a_j	b_j	c_j	d_j
0	0.90	1.30	0.54	0.00	-0.25
1	1.30	1.50	0.42	-0.30	0.95
2	1.90	1.85	1.09	1.41	-2.96
3	2.10	2.10	1.29	-0.37	-0.45
4	2.60	2.60	0.59	-1.04	0.45
5	3.00	2.70	-0.02	-0.50	0.17
6	3.90	2.40	-0.50	-0.03	0.08
7	4.40	2.15	-0.48	0.08	1.31
8	4.70	2.05	-0.07	1.27	-1.58
9	5.00	2.10	0.26	-0.16	0.04
10	6.00	2.25	0.08	-0.03	-0.00
11	7.00	2.30	0.01	-0.04	-0.02
12	8.00	2.25	-0.14	-0.11	0.02
13	9.20	1.95	-0.34	-0.05	-0.01
14	10.50	1.40	-0.53	-0.10	-0.02
15	11.30	0.90	-0.73	-0.15	1.21
16	11.60	0.70	-0.49	0.94	-0.84
17	12.00	0.60	-0.14	-0.06	0.04
18	12.60	0.50	-0.18	0.00	-0.45
19	13.00	0.40	-0.39	-0.54	0.60
20	13.30	0.00	0.00	0.00	0.00

Con estos datos podemos construir los splines de cada sección, los cuales junto con los puntos originales se ven de la siguiente forma



3. El código utilizado para el spline cúbico condicionado es el siguiente

```
double spline_condicionado(int N, double *x, double *y, double *puntos_condicionados, double *
    valores_condicionados, double punto) {
    double *a = (double *)malloc(N * sizeof(double));
    double *b = (double *)malloc(N * sizeof(double));
    double *c = (double *)malloc(N * sizeof(double));
    double *d = (double *)malloc(N * sizeof(double));

    // Calcular los coeficientes 'h' y 'alpha'
    double *h = (double *)malloc((N - 1) * sizeof(double));
    double *alpha = (double *)malloc((N - 1) * sizeof(double));

    for (int i = 0; i < N - 1; i++) {
        h[i] = x[i + 1] - x[i];
        alpha[i] = (3.0 / h[i]) * (y[i + 1] - y[i]) - (3.0 / (i == 0 ? h[i] : h[i - 1])) * (y[i] -
     y[i - 1]);

        // Aplicar las condiciones en los puntos dados
        for (int j = 0; j < N - 1; j++) {
            if (x[i] == puntos_condicionados[j]) {
                alpha[i] = valores_condicionados[j];
            }
            if (x[i + 1] == puntos_condicionados[j]) {
                alpha[i] = valores_condicionados[j];
            }
        }
    }

    // Resto del c digo sin cambios
    double *l = (double *)malloc(N * sizeof(double));
    double *u = (double *)malloc(N * sizeof(double));
    double *w = (double *)malloc(N * sizeof(double));

    l[0] = 1.0;
    u[0] = 0.0;
    w[0] = 0.0;

    for (int i = 1; i < N - 1; i++) {
        l[i] = 2 * (h[i - 1] + h[i]) - h[i - 1] * u[i - 1];
        u[i] = h[i] / l[i];
        w[i] = (alpha[i] - alpha[i - 1]) - h[i - 1] * w[i - 1];
        w[i] /= l[i];
    }

    l[N - 1] = 1.0;
    u[N - 1] = 0.0;
    w[N - 1] = 0.0;

    // Calcular los coeficientes c, b, d y a
    for (int i = N - 2; i >= 0; i--) {
        c[i] = w[i] - u[i] * c[i + 1];
        b[i] = (y[i + 1] - y[i]) / h[i] - h[i] * (c[i + 1] + 2 * c[i]) / 3.0;
        d[i] = (c[i + 1] - c[i]) / (3.0 * h[i]);
        a[i] = y[i];
    }

    // Encontrar el intervalo correspondiente al punto
    int intervalo = 0;
    for (int i = 0; i < N - 1; i++) {
        if (x[i] <= punto && punto <= x[i + 1]) {
            intervalo = i;
            break;
        }
    }

    // Calcular el valor del spline en el punto
    double dx = punto - x[intervalo];
    double resultado = a[intervalo] + b[intervalo] * dx + c[intervalo] * dx * dx + d[intervalo] *
    dx * dx * dx;

    // Liberar la memoria
    free(a);
    free(b);
    free(c);
    free(d);
    free(h);
    free(alpha);
    free(l);
    free(u);
```

```
    free(w);

    return resultado;
}
```

Al igual que en el problema anterior, se puede utilizar modificar el siguiente código para almacenar los valores de a_j, b_j, c_j y d_j.Los datos para cada una de las tres curvas son los siguientes

i	$x[i]$	$a[i]$	$b[i]$	$c[i]$	$d[i]$
0	1.000000	3.000000	0.868836	0.000000	-0.168836
1	2.000000	3.700000	0.695662	-0.506507	0.098947
2	5.000000	3.900000	-0.305136	0.384019	0.221118
3	6.000000	4.200000	1.359588	1.047372	-0.906960
4	7.000000	5.700000	1.933453	-1.673507	0.640055
5	8.000000	6.600000	-0.093398	0.246657	-0.037479
6	10.000000	7.100000	-0.206518	0.021783	0.000871
7	13.000000	6.700000	-0.628981	0.029618	-0.002468
8	17.000000	0.000000	0.000000	0.000000	0.000000

Cuadro 1: Datos spline cúbico fijo para curva 1

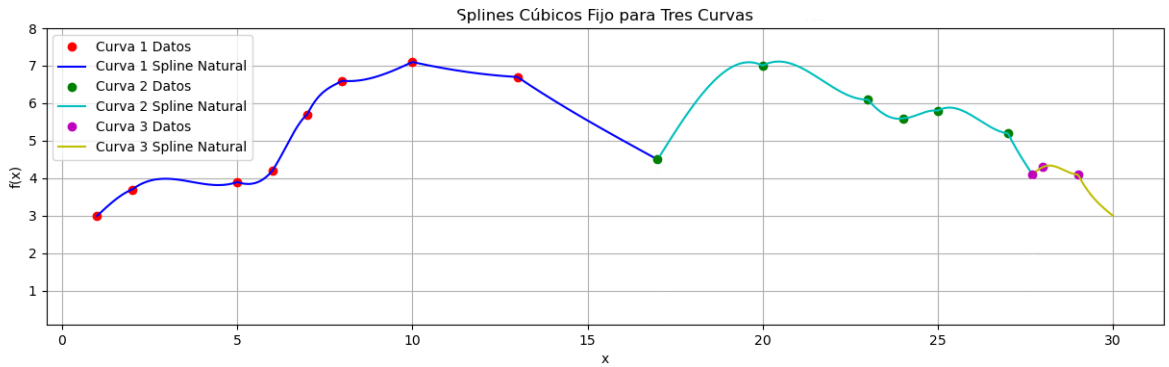
i	$x[i]$	$a[i]$	$b[i]$	$c[i]$	$d[i]$
0	17.000000	4.500000	1.491315	0.000000	-0.073109
1	20.000000	7.000000	0.517371	-0.657981	0.128508
2	23.000000	6.100000	-1.094132	0.498591	0.095540
3	24.000000	5.600000	-0.010328	0.785213	-0.574885
4	25.000000	5.800000	0.535443	-0.939442	0.260860
5	27.000000	5.200000	-1.863431	0.625719	-0.297962
6	27.700000	0.000000	0.000000	0.000000	0.000000

Cuadro 2: Datos spline cúbico fijo para curva 2

i	$x[i]$	$a[i]$	$b[i]$	$c[i]$	$d[i]$
0	27.700000	4.100000	0.819007	0.000000	-1.692671
1	28.000000	4.300000	0.471986	-1.523404	0.851418
2	29.000000	4.100000	-1.787234	1.030851	-0.343617
3	30.000000	0.000000	0.000000	0.000000	0.000000

Cuadro 3: Datos spline cúbico fijo para curva 3

Con estos valores se pueden construir la siguiente gráfica



4. Conclusiones

Podemos observar que la utilización de los diferentes métodos nos permite tener un mayor control acerca del comportamiento general del modelo que estamos creando, en particular cuando se realizó la interpolación de Hermite se puede ver que este es mucho más sensible (por lo tanto, con mayor potencial a ser más exacto por esta sensibilidad local que es una especie de curvatura local) para un redondeo o truncamiento de datos, esto debido también a que contamos con la información de la derivada de ellos.

Por otro lado, podemos ver que se tiene una mayor suavidad al momento de construir los datos así estabilidad al momento de realizar el modelo, aunque esta suavidad puede resultar contraproducente pues significa sacrificar información del sistema tal y como es las derivadas en uno o más puntos.

En general la decisión de cuál tomar dependerá de la naturaleza de los datos y la naturaleza del modelo que queramos construir: deseamos uno donde se incluya información fuerte como la pendiente en cada punto pero por ente inestable numéricamente, o uno que nos proporcione un panorama más general del comportamiento pero sacrificando información necesaria (en el caso de que lo haya o a veces suponiendo información inaccesible) para comprender completamente el sistema.

5. Referencias

1. Kong, Q., Siauw, T., & Bayen, A. (2020). Python programming and numerical methods: A guide for engineers and scientists. Academic Press.
2. Richard. L. Burden y J. Douglas Faires, Análisis Numérico, 7a Edición, Edito-rial Thomson Learning, 2002.
3. Samuel S M Wong, Computational Methods in Physics and Enginering, Ed. World Scientific, 3rd Edition, 1997.
4. Teukolsky, S. A., Flannery, B. P., Press, W. H., & Vetterling, W. T. (1992). Numerical recipes in C. SMR, 693(1), 59-70.
5. William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery, Numerical Recipes: The Art of Scientific Computing, 3rd Edition, Cambridge University Press, 2007