

Metoda Neville. Metode de interpolare cu funcții spline cubice. Curbe Bézier. Algoritmul De Casteljau

Colaboratori: Andrei STAN, Dumitru-Clementin Cercel, Bogdan Marchiș

February 17, 2025

Cuprins

1	Obiective laborator	1
2	Noțiuni teoretice	1
2.1	Metoda Neville	1
2.2	Metode de interpolare cu funcții spline cubice	2
2.2.1	Interpolare cu funcții spline în clasa C^1 :	2
2.2.2	Interpolare cu funcții spline în clasă C^2 :	3
2.3	Curbe Bézier	4
2.4	Algoritmul De Casteljau	5
3	Probleme rezolvate	6
3.1	Problema 1	6
3.2	Problema 2	7
3.3	Problema 3	7
3.4	Problema 4	8
3.5	Problema 5	10
4	Probleme propuse	11
4.1	Problema 1	11
4.2	Problema 2	11
4.3	Problema 3	12

1 Obiective laborator

În urma parcurgerii acestui laborator, studentul va fi capabil să:

- calculeze valoarea unei funcții într-un punct folosind metoda Neville;
- aproximeze traiectoria unei funcții cu ajutorul funcțiilor spline cubice;
- traseze o curba Bézier folosind algoritmul De Casteljau.

2 Noțiuni teoretice

2.1 Metoda Neville

Se consideră o funcție $f : [a, b] \rightarrow R$ cunoscută într-o mulțime finită de puncte x_0, x_1, \dots, x_n (numite suportul interpolării) prin valorile:

$$f(x_0), f(x_1), \dots, f(x_n).$$

Metoda Neville este o metodă de interpolare care aproximează comportamentul funcției f în afara acestor $n + 1$ puncte. În cele ce urmează, folosim notația $P_{ij}(x)$ pentru a reprezenta polinomul de interpolare de grad $j - i$ care trece prin punctele $(x_l, f(x_l)), l = i, i + 1, \dots, j$.

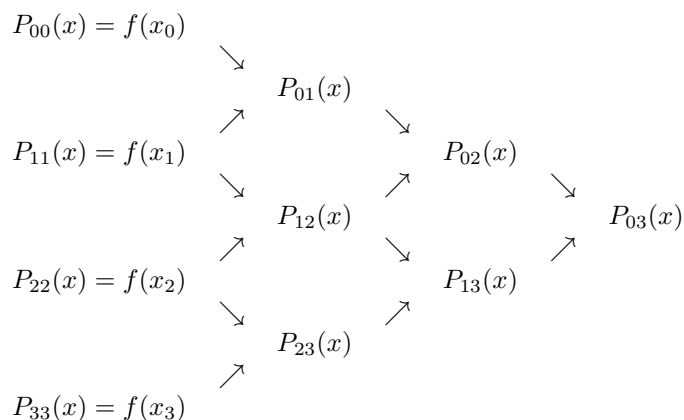
Polinomul de interpolare P_{ij} este dat de relație de recurență:

$$P_{ij}(x) = \frac{x - x_j}{x_i - x_j} P_{i,j-1}(x) + \frac{x_i - x}{x_i - x_j} P_{i+1,j}(x), \quad 0 \leq i < j \leq n$$

unde

$$P_{ii}(x) = f(x_i), \quad i = 0 : n.$$

În prima iterație, metoda Neville construiește polinoame de interpolare de grad 0, reprezentate prin $P_{ii}(x), i = 0 : n$. În următoarea iterație, oricare două polinoame de interpolare de grad 0, alăturate, P_{ii} și $P_{i+1,i+1}, i = 0 : n - 1$, formează un polinom de interpolare de grad 1 notat cu $P_{i,i+1}(x)$. Procesul continuă până când se obține polinomul de interpolare $P_{0,n}$ care trece prin toate cele $n+1$ puncte $(x_l, f(x_l)), l = 0, \dots, n$. De exemplu, procesul iterativ de obținere al polinomului de interpolare $P_{03}(x)$, pentru $n = 3$, este arătat în schema următoare:



2.2 Metode de interpolare cu funcții spline cubice

În continuare, considerăm că funcția $f : [a, b] \rightarrow R$ este cunoscută atât într-o mulțime finită de puncte x_0, x_1, \dots, x_n prin valorile:

$$f(x_0), f(x_1), \dots, f(x_n)$$

cât și prin derivatele de ordin I ale funcției în aceste puncte:

$$f'(x_0), f'(x_1), \dots, f'(x_n)$$

În cazul interpolării cu funcții spline cubice, comportamentul funcției f se va studia local pe subintervalele:

$$[x_0, x_1), [x_1, x_2), \dots, [x_i, x_{i+1}), \dots, [x_{n-1}, x_n)$$

Astfel, pentru fiecare subinterval $[x_i, x_{i+1})$ se va considera o funcție de interpolare de gradul 3 (numită funcție spline cubică) de forma:

$$s_i : [x_i, x_{i+1}) \rightarrow R, s_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$$

Prin schimbarea de variabilă $t = \frac{x - x_i}{x_{i+1} - x_i}$ se obține următoarea formă parametrică pentru o funcție spline cubică:

$$s_i(t) = a_i + b_i h_i t + c_i h_i t^2 + d_i h_i t^3$$

unde am notat $h_i = x_{i+1} - x_i$.

În general, se folosește baza de interpolare Bernstein pentru a eficientiza procesul de calculare al coeficienților funcțiilor spline cubice:

$$(1 - t)^3, 3t(1 - t)^2, 3t^2(1 - t), t^3$$

Folosind baza de interpolare Bernstein, funcția spline cubică devine :

$$s_i(t) = a'_i(1 - t)^3 + 3b'_i t(1 - t)^2 + 3c'_i t^2(1 - t) + d'_i t^3$$

2.2.1 Interpolare cu funcții spline în clasa C^1 :

Pentru a determina cei $4n$ coeficienți, se impun următoarele condiții în cazul interpolării cu funcții spline în clasă C^1 :

- $2n + 2$ condiții de interpolare de tip Hermite:

$$s_i(x_i) = f(x_i), \quad i = 0 : n - 1$$

$$s_{n-1}(x_n) = f(x_n)$$

$$s'_i(x_i) = f'(x_i), \quad i = 0 : n - 1$$

$$s'_{n-1}(x_n) = f'(x_n)$$

- $2n - 2$ condiții de racordare ce asigură continuitatea și derivabilitatea funcțiilor spline cubice vecine:

$$s_i(x_{i+1}) = s_{i+1}(x_{i+1}), \quad i = 0 : n - 2$$

$$s'_i(x_{i+1}) = s'_{i+1}(x_{i+1}), \quad i = 0 : n - 2$$

În rezolvarea sistemului format din cele $4n$ ecuații, considerăm pentru funcțiile spline cubice forma parametrică în baza Bernstein. Se obțin următoarele relații pentru coeficienți:

$$\begin{aligned}a_i' &= f(x_i), \quad i = 0 : n - 1 \\d_i' &= f(x_{i+1}), \quad i = 0 : n - 1 \\b_i' &= f(x_i) + \frac{h_i}{3} f'(x_i), \quad i = 0 : n - 1 \\c_i' &= f(x_{i+1}) - \frac{h_i}{3} f'(x_{i+1}), \quad i = 0 : n - 1\end{aligned}$$

iar forma parametrică pentru funcțiile spline cubice în baza Bernstein devine:

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

În cele din urmă, se poate obține forma în variabila x pentru fiecare funcție spline cubică în clasă C^1 prin schimbarea de variabilă $t = \frac{x-x_i}{h_i}$.

2.2.2 Interpolare cu funcții spline în clasă C^2 :

În cazul interpolării cu funcții spline în clasă C^2 , cei $4n$ coeficienți se determină prin impunerea a:

- $n + 1$ condiții de interpolare de tip Lagrange:

$$s_i(x_i) = f(x_i), \quad i = 0 : n - 1$$

$$s_{n-1}(x_n) = f(x_n)$$

- $3n - 3$ condiții de continuitate, derivabilitate și curbură pentru funcțiile spline cubice vecine:

$$s_i(x_{i+1}) = s_{i+1}(x_{i+1}), \quad i = 0 : n - 2$$

$$s_i'(x_{i+1}) = s_{i+1}'(x_{i+1}), \quad i = 0 : n - 2$$

$$s_i''(x_{i+1}) = s_{i+1}''(x_{i+1}), \quad i = 0 : n - 2$$

- următoarelor 2 condiții pentru funcțiile spline naturale:

$$s_0''(x_0) = 0$$

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

respectiv următoarelor 2 condiții pentru funcțiile spline tensionate:

$$s_0'(x_0) = f'(x_0)$$

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

În cele ce urmează, se consideră pentru fiecare funcție spline cubică forma în variabila x . Coeficienții funcțiilor spline cubice în clasă C^2 sunt dați de relațiile:

$$\begin{aligned} a_i &= f(x_i), \quad i = 0 : n \\ d_i &= \frac{c_{i+1} - c_i}{3h_i}, \quad i = 0 : n - 1 \\ b_i &= \frac{a_{i+1} - a_i}{h_i} - \frac{h_i}{3}(2c_i + c_{i+1}), \quad i = 0 : n - 1 \end{aligned}$$

iar coeficienții $c_i, i = 0 : n - 1$ se obțin prin rezolvarea unui sistem tridiagonal de forma:

- pentru funcții spline naturale

$$\begin{bmatrix} 1 & 0 & 0 & & 0 \\ h_0 & 2(h_0 + h_1) & h_1 & & 0 \\ & \ddots & \ddots & \ddots & \\ & & h_{n-2} & 2(h_{n-2} + h_{n-1}) & h_{n-1} \\ 0 & & & 0 & 1 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \\ c_n \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{3(a_2 - a_1)}{h_1} - \frac{3(a_1 - a_0)}{h_0} \\ \vdots \\ \frac{3(a_n - a_{n-1})}{h_{n-1}} - \frac{3(a_{n-1} - a_{n-2})}{h_{n-2}} \\ 0 \end{bmatrix}$$

- pentru funcții spline tensionate

$$\begin{bmatrix} 2h_0 & h_0 & 0 & & 0 \\ h_0 & 2(h_0 + h_1) & h_1 & & 0 \\ & \ddots & \ddots & \ddots & \\ & & h_{n-2} & 2(h_{n-2} + h_{n-1}) & h_{n-1} \\ 0 & & & h_{n-1} & 2h_{n-1} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \\ c_n \end{bmatrix} = \begin{bmatrix} \frac{3(a_1 - a_0)}{h_0} - 3f'(x_0) \\ \frac{3(a_2 - a_1)}{h_1} - \frac{3(a_1 - a_0)}{h_0} \\ \vdots \\ \frac{3(a_n - a_{n-1})}{h_{n-1}} - \frac{3(a_{n-1} - a_{n-2})}{h_{n-2}} \\ 3f'(x_n) - \frac{3(a_n - a_{n-1})}{h_{n-1}} \end{bmatrix}$$

Funcția spline s_n a fost introdusă pentru a ajuta la calcularea funcțiilor spline $s_i, i = 0 : n - 1$.

2.3 Curbe Bézier

Fiind dată o mulțime de $n + 1$ puncte P_0, P_1, \dots, P_n (numite *puncte de control*) în plan sau în spațiu, curba Bézier de grad n determinată de aceste puncte are forma parametrică:

$$B(t) = \sum_{i=0}^n P_i B_{i,n}(t), \quad t \in [0, 1]$$

unde $B_{i,n}(t)$ se numesc polinoame Bernstein de grad n definite prin relația:

$$B_{i,n}(t) = \binom{n}{i} (1-t)^{n-i} t^i, \quad i = 0 : n$$

Gradul polinomului care aproximează curba Bézier depinde de numărul punctelor de control. Fiecare punct de control afectează forma curbei Bézier în mod particular. Dacă modificăm poziția unui punct de control și forma curbei Bézier va fi afectată.

Spre deosebire de funcțiile spline, curba Bézier nu trece prin fiecare punct de control. Orice curbă Bézier începe în punctul de control P_0 și se termină în punctul de control P_n adică:

$$B(0) = P_0, B(1) = P_n$$

Mai mult, curba Bézier este tangentă segmentelor P_0P_1 și $P_{n-1}P_n$ adică:

$$B'(0) = n(B_1 - B_0), B'(1) = n(B_n - B_{n-1})$$

O altă proprietate este aceea că orice curbă Bézier este conținută complet de înfășurătoarea convexă pe care punctele de control o definesc.

În mod uzual, se folosesc curbe Bézier cubice determinate de 4 puncte de control, având forma parametrică:

$$B(t) = P_0B_{0,3}(t) + P_1B_{1,3}(t) + P_2B_{2,3}(t) + P_3B_{3,3}(t) = P_0(1-t)^3 + 3P_1t(1-t)^2 + 3P_2t^2(1-t) + P_3t^3 =$$

$$\begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}, \quad t \in [0, 1]$$

2.4 Algoritmul De Casteljau

Într-o abordare directă, calcularea unui punct aflat pe o curbă Bézier se obține folosind ecuația parametrică: $B(t) = \sum_{i=0}^n P_i B_i^n(t), t \in [0, 1]$. Această metodă este inefficientă deoarece ridică numere mici la puteri mari, generând astfel erori mari.

Algoritmului De Casteljau este o modalitate mult mai eficientă de calculare a unui punct aflat pe o curbă Bézier. Acest algoritm este puțin mai lent dar este numeric stabil și cu ajutorul său putem obține detalii despre curba Bézier:

- vectorul tangent într-un punct de pe curba Bézier folosind calculul derivatei;
- divizarea curbei Bézier. Uneori este necesar să separăm o curbă Bézier în alte două curbe Bézier.

Algoritmul De Casteljau folosește relația de recurență următoare:

$$P_i^{(0)} = P_i, \quad i = 0 : n$$

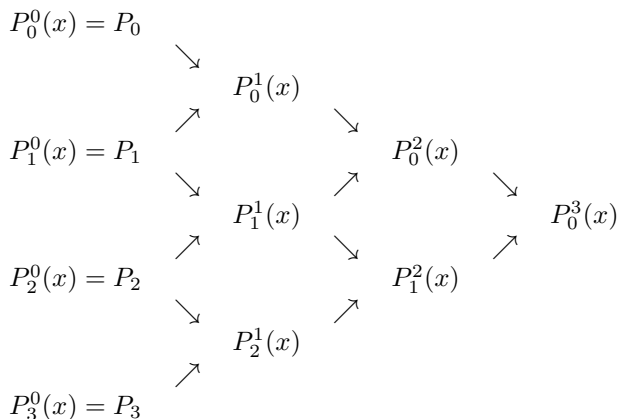
$$P_i^{(j)} = P_i^{(j-1)}(1 - t_0) + P_{i+1}^{(j-1)}t_0, \quad i = 0 : n - j, j = 1 : n$$

unde $B(t_0) = P_0^{(n)}$.

Algoritmul De Casteljau poate fi considerat o interpolare liniară repetată. Fie $P_i^{(0)}$ și $P_{i+1}^{(0)}$ două puncte de control succesive și $P_i^{(1)}$ un punct care împarte segmentul $P_i^{(0)}P_{i+1}^{(0)}$ în raportul $t/(1-t)$. Putem scrie următoarea relație pentru $P_i^{(1)}$:

$$P_i^1 = P_i^0 + t(P_{i+1}^0 - P_i^0) = (1-t)P_i^0 + tP_{i+1}^0$$

Se formează în acest fel poligonul $P_0^1, P_1^1, \dots, P_{n-1}^1$. Se aplică relația de recurență noului poligon obținându-se poligonul $P_0^2, P_1^2, \dots, P_{n-2}^2$. Repetând procesul de n ori se obține un singur punct P_0^n . Acest punct obținut se află pe curba Bézier. De exemplu, procesul iterativ de obținere al polinomului de interpolare $P_{03}(x)$, pentru $n = 3$, este arătat în schema următoare:



3 Probleme rezolvate

3.1 Problema 1

Se consideră funcția f cunoscută prin valorile:

x	0.4	0.9	1.5	2.5
$f(x)$	0.9	0.5	0.2	0.1

Determinați valoarea funcției în punctul $x = 4$.

Soluție:

$$P_{00}(4) = f(x_0) = 0.9$$

$$P_{11}(4) = f(x_1) = 0.5$$

$$P_{22}(4) = f(x_2) = 0.2$$

$$P_{33}(4) = f(x_3) = 0.1$$

$$P_{01}(4) = \frac{(4 - x_1)P_{00}(4) + (x_0 - 4)P_{11}(4)}{x_0 - x_1} = \frac{(4 - 0.9)0.9 + (0.4 - 4)0.5}{0.4 - 0.9} = -1.98$$

$$P_{12}(4) = \frac{(4 - x_2)P_{11}(4) + (x_1 - 4)P_{22}(4)}{x_1 - x_2} = \frac{(4 - 1.5)0.5 + (0.9 - 4)0.2}{0.9 - 1.5} = -1.05$$

$$P_{23}(4) = \frac{(4 - x_3)P_{22}(4) + (x_2 - 4)P_{33}(4)}{x_2 - x_3} = \frac{(4 - 2.5)0.2 + (1.5 - 4)0.1}{1.5 - 2.5} = -0.05$$

$$P_{02}(4) = \frac{(4 - x_2)P_{01}(4) + (x_0 - 4)P_{12}(4)}{x_0 - x_2} = \frac{(4 - 1.5)(-1.98) + (0.4 - 4)(-1.05)}{0.4 - 1.5} = 1.06$$

$$P_{13}(4) = \frac{(4 - x_3)P_{12}(4) + (x_1 - 4)P_{23}(4)}{x_1 - x_3} = \frac{(4 - 2.5)(-1.05) + (0.9 - 4)(-0.05)}{0.9 - 1.5} = 0.89$$

$$P_{03}(4) = \frac{(4 - x_3)P_{02}(4) + (x_0 - 4)P_{13}(4)}{x_0 - x_3} = \frac{(4 - 2.5)1.06 + (0.4 - 4)0.89}{0.4 - 2.5} = 0.76$$

Prin urmare, am obținut $f(4) \simeq P_{03}(4) = 0.76$

3.2 Problema 2

Scrieți un program OCTAVE care calculează valoarea unei funcții într-un punct folosind metoda Neville. Programul primește ca parametri de intrare: x - suportul interpolării, y - vectorul ordonatelor pentru valorile din suportul interpolării, xi - abscisa în care se calculează valoarea funcției. Rezultatul programului este valoarea funcției în xi .

Soluție:

```
function yi = Neville(x, y, xi)
    n = length(x);

    for k = 1 : n-1
        for i = 1 : n-k
            raport = (xi-x(k+i))/(x(i)-x(k+i));
            y(i) = raport*y(i)+(1-raport)*y(i+1);
        endfor
    endfor

    yi = y(1);
endfunction
```

Listing 1: Metoda Neville de interpolare.

3.3 Problema 3

Scrieți un program OCTAVE care calculează valoarea unei funcții într-un punct, ca rezultat al interpolării obținute folosind funcții spline în clasă C^1 . Programul primește ca parametri de intrare: x - suportul interpolării, y - vectorul ordonatelor pentru valorile din suportul interpolării, dy - vectorul derivatelor pentru valorile din suportul interpolării, xi - abscisa în care se calculează valoarea funcției. Rezultatul programului este valoarea funcției în xi .

Soluție:

```
function yi = SplineC1(x, y, dy, xi)
    n = length(x);

    for i = 1 : n-1
        a(i) = y(i);
        d(i) = y(i+1);
        h(i) = x(i+1)-x(i);
        b(i) = y(i)+h(i)*dy(i)/3;
        c(i) = y(i+1)-h(i)*dy(i+1)/3;
    endfor

    for i = 2 : n
        if xi <= x(i)
            k = i-1;
            break;
        endif
    endfor

    t = (xi-x(k))/h(k);
    yi = (a(k)*(1-t)^3)+(3*b(k)*t*(1-t)^2)+(3*c(k)*t^2*(1-t))+(d(k)*t^3);
endfunction
```

Listing 2: Interpolare cu funcții spline în clasă C^1 .

Pentru testarea programului, puteți folosi următoarea secvență:

```
x = 0:pi/3:7*pi;
y = sin(x)+cos(3*x);
dy = cos(x)-3*sin(3*x);
xi = 0:pi/15:7*pi;

for i = 1 : length(xi)
    yi(i) = feval('SplineC1', x, y, dy, xi(i));
endfor

plot(x, y, 'g-o', xi, yi, 'r-*');
legend('data', 'spline C1');
axis([0 23 -2 2.7]);
```

Listing 3: Fișier de testare.

iar rezultatul este:

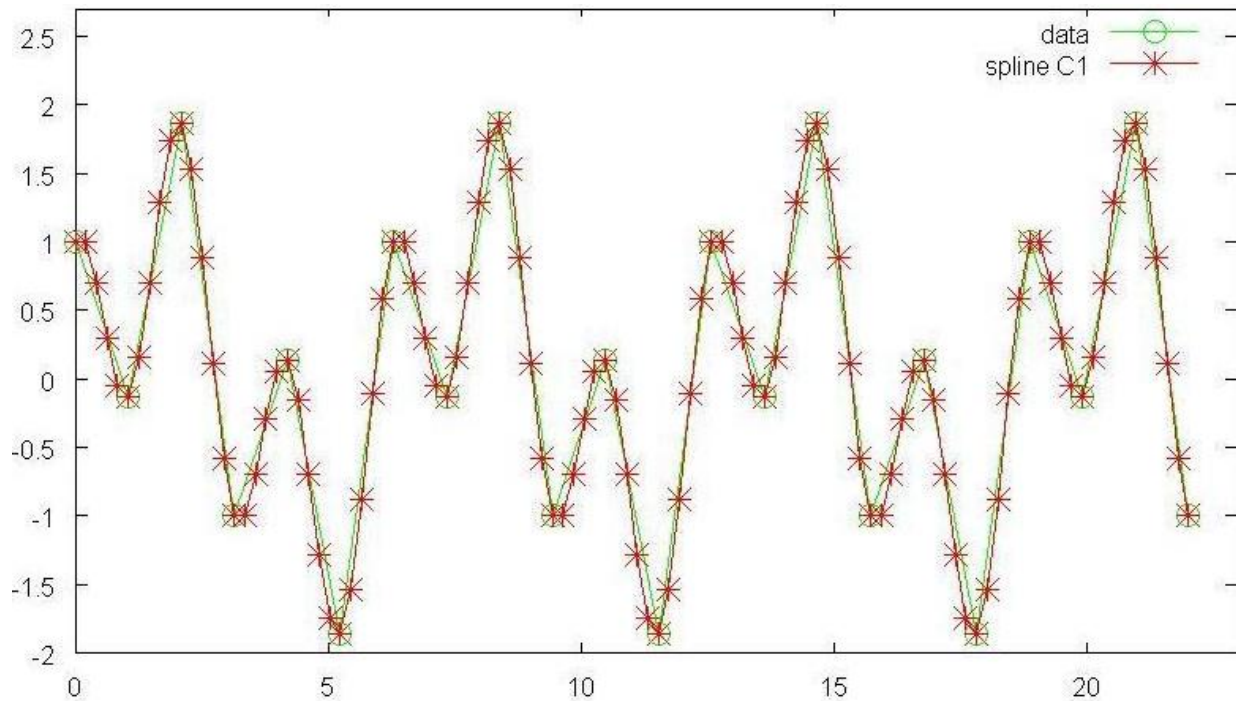


Figura 1: Graficul rezultat pentru fișierul de testare.

3.4 Problema 4

Scrieți un program OCTAVE care calculează valoarea unei funcții într-un punct, ca rezultat al interpolării obținute folosind funcții spline naturale în clasă C^2 . Programul primește ca parametri de intrare: x - suportul interpolării, y - vectorul ordonatelor pentru valorile din suportul interpolării, xi - abcisa în care se calculează valoarea funcției. Rezultatul programului este valoarea funcției în xi .

Soluție:

```
function yi = SplineC2natural(x, y, xi)
```

```

n = length(x);

h(1:n-1) = x(2:n)-x(1:n-1);
a(1:n) = y(1:n);

% aa - vectorul subdiagonala
aa(1:n-2) = h(1:n-2);
aa = [aa 0];

% bb - vectorul diagonala principala
bb(1) = 1;
bb(n) = 1;
for i = 2 : n-1
    bb(i) = 2*(h(i-1)+h(i));
end

% cc - vectorul supradiagonala
cc(1) = 0;
cc(2:n-1) = h(2:n-1);

% g - vectorul termenilor liberi
g(1) = 0;
g(n) = 0;
for i = 2 : n-1
    g(i) = 3*(a(i+1)-a(i))/h(i)-3*(a(i)-a(i-1))/h(i-1);
endfor

% rezolvare sistem tridiagonal
c = Thomas(aa, bb, cc, g);

for i = 1 : n-1
    d(i) = (c(i+1)-c(i))/(3*h(i));
    b(i) = (a(i+1)-a(i))/h(i)-((2*c(i)+c(i+1))*h(i))/3;
endfor

for i = 2 : n
    if xi <= x(i)
        k = i-1;
        break;
    endif
endfor

yi = a(k)+(b(k)*(xi-x(k)))+(c(k)*(xi-x(k))^2)+(d(k)*(xi-x(k))^3);
end

```

Listing 4: Interpolare cu funcții spline naturale în clasă C^2 .

Pentru testarea programului puteți folosi următoarea secvență:

```

x = 0:pi/3:7*pi;
y = sin(x)+cos(3*x);
xi = 0:pi/15:7*pi;

for i = 1 : length(xi)
    yi(i) = feval('SplineC2natural', x, y, xi(i));
endfor

plot(x, y, 'g-o', xi, yi, 'r-*');

```

```

legend('data', 'spline C2 natural');
axis([0 23 -2 2.7]);

```

Listing 5: Fișier de testare.

iar rezultatul este:

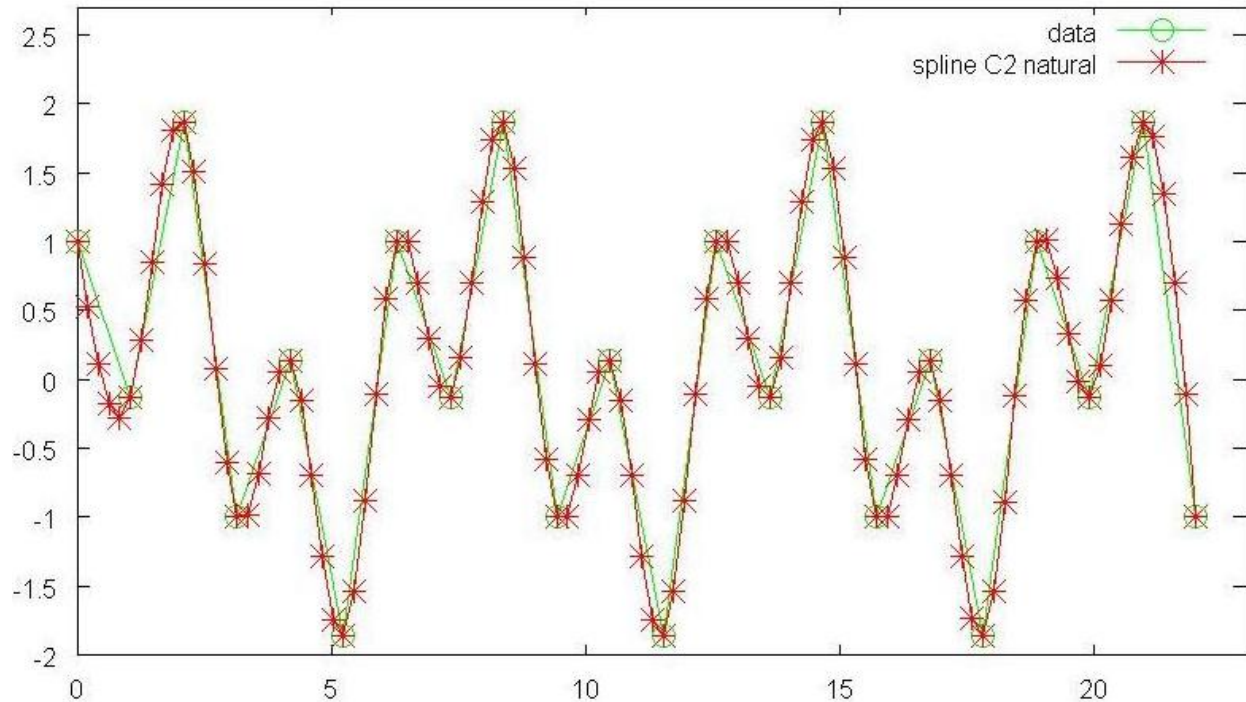


Figura 2: Graficul rezultat pentru fișierul de testare.

3.5 Problema 5

Scrieți o funcție OCTAVE care să calculeze un punct aflat pe o curbă Bézier folosind algoritmului De Casteljau. Funcția primește ca parametri de intrare: x - vectorul absciselor punctelor de control, y - vectorul ordonatelor punctelor de control, t - parametru. Rezultatul funcției este un punct P aflat pe curba Bézier.

Soluție:

```

function P = Casteljau(x, y, t)
    n = length(x);

    plot(x, y, 'k-o');
    legend('puncte de control');
    hold on;
    axis([0.5, 4.5, 0.2, 2.1]);

    for j = 1 : n-1
        for i = 1 : n-j
            x(i) = (1-t)*x(i)+t*x(i+1);
            y(i) = (1-t)*y(i)+t*y(i+1);
        endfor
    endfor

```

```

    plot(x(1:n-j), y(1:n-j), 'r-*');
endfor

P = [x(1) y(1)];
endfunction

```

Listing 6: Algoritmului De Casteljaou.

Pentru testarea funcției anterioare puteți folosi următoarea secvență care trasează o curbă Bézier folosind algoritmul De Casteljaou:

```

x = [1 0.7 2.7 3.7];
y = [0.5 2 2 0.5];

i = 1;
B_x = zeros();
B_y = zeros();
hold off;

for t = 0 : 0.05 : 1
    P = Casteljaou(x, y, t);
    B_x(i) = P(1);
    B_y(i) = P(2);
    plot(B_x, B_y, 'b-s');
    legend('puncte calculate', 'puncte aflate pe curba Bezier');
    i++;

    pause(0.5);
    hold off;
endfor

```

Listing 7: Fișier de testare.

iar rezultatul este:

4 Probleme propuse

4.1 Problema 1

Se consideră funcția f cunoscută prin valorile:

x	-2	2	3	5
$f(x)$	-3	1	3	7

Determinați valoarea funcției în punctul $x = 1$.

4.2 Problema 2

Scrieți un program OCTAVE care calculează valoarea unei funcții într-un punct ca rezultat al interpolării obținute folosind funcții spline tensionate în clasă C^2 . Programul primește ca parametri de intrare: x - suportul interpolării, y - vectorul ordonatelor pentru valorile din suportul interpolării, $dy1$ - derivata funcției în primul punct din suportul interpolării, dyn - derivata funcției în ultimul punct din suportul interpolării, xi - abscisa în care se calculează valoarea funcției. Rezultatul programului este valoarea funcției în xi .

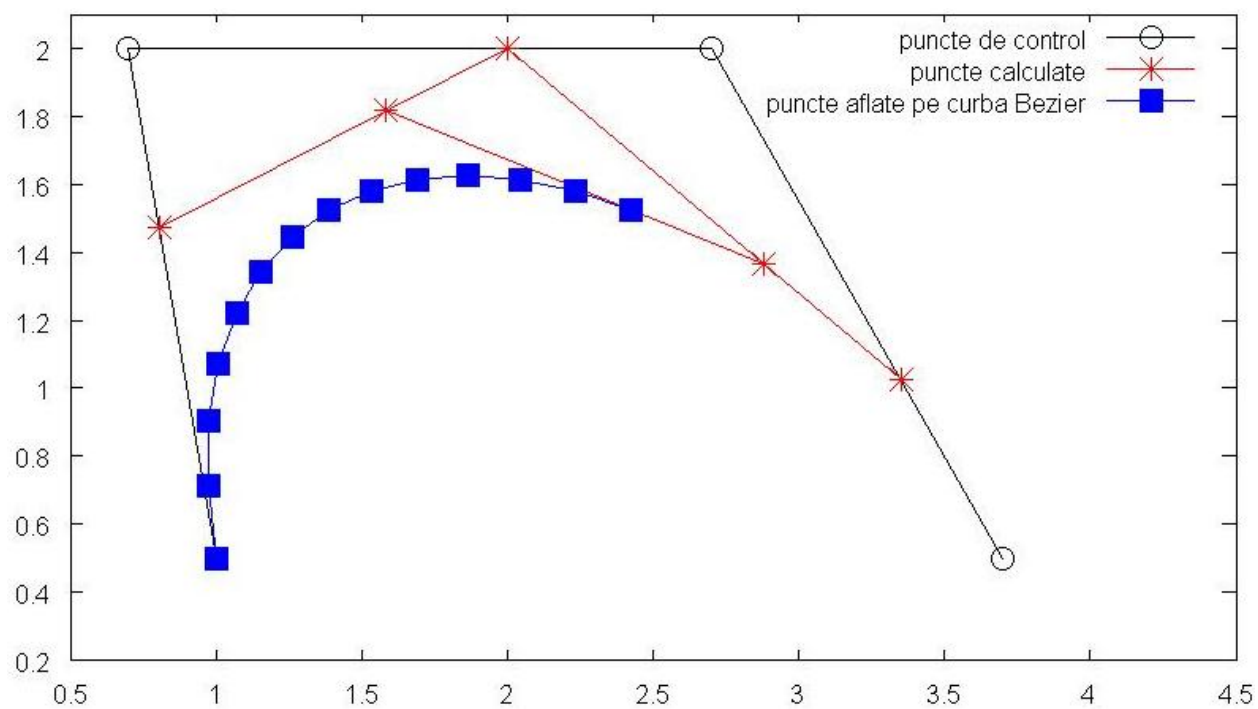


Figura 3: Graficul intermediar rezultat pentru fișierul de testare.

4.3 Problema 3

Se consideră funcția f cunoscută prin valorile:

x	-3	1	2
$f(x)$	0	3	4
$f'(x)$	1	2	3

Determinați funcțiile spline în clasa C^1 pentru interpolarea lui f .

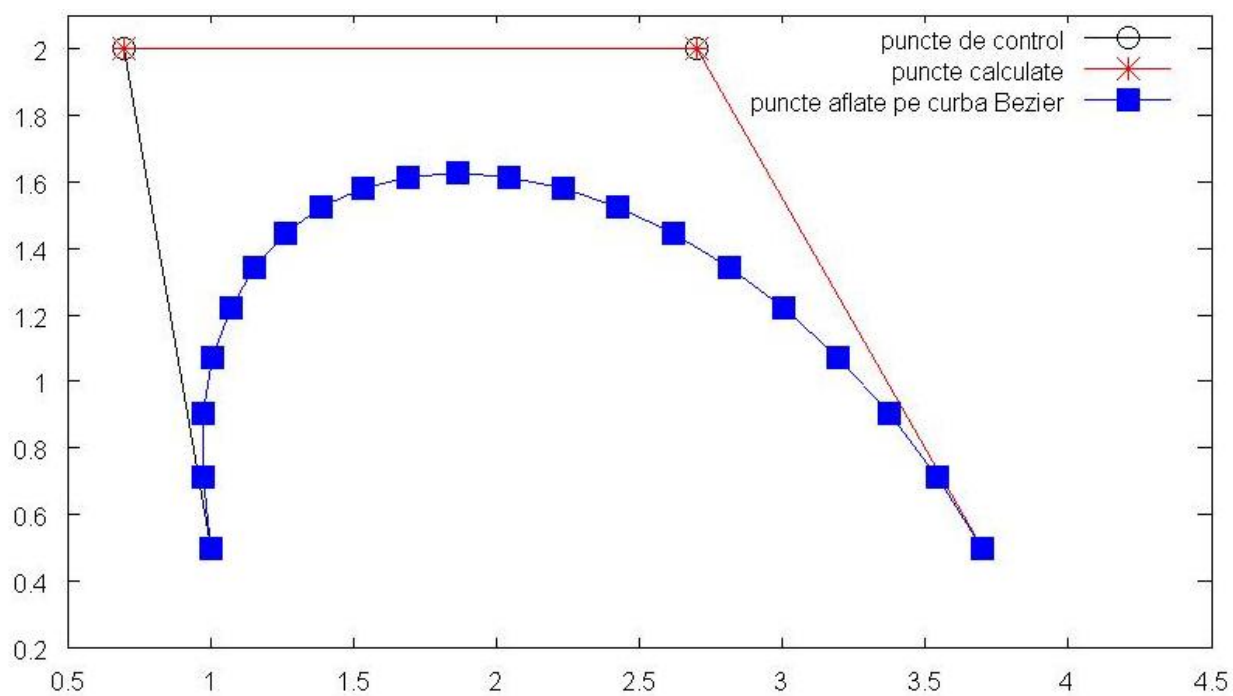


Figura 4: Graficul final rezultat pentru fișierul de testare.