Obliczanie wartości i współczynników naturalnej funkcji sklejenia stopnia 3.

1. Zastosowanie  
   Program oblicza współczynniki naturalnej funkcji sklejenia stopnia 3 dla danych punktów (x, y) oraz wyznacza wartość funkjcji w zadanym punkcie xi w trybach:

* 1: arytmetyki zmiennoprzecinkowej,
* 2: arytmetyki przedziałowej (konwersja pojedynczych wartości na przedziały),
* 3: arytmetyki przedziałowej (jawne granice przedziału).

1. Opis metody  
   Metoda naturalnego funkcji sklejenia stopnia 3 polega na znalezieniu ciągłych funkcji wielomianowych trzeciego stopnia między kolejnymi węzłami tak, aby druga pochodna na krańcach była zerowa. Układ równań tridiagonalnych dla drugich pochodnych jest rozwiązywany metodą eliminacji Gaussa w arytmetyce stosownej dla wybranego trybu.
2. Wywołanie funkcji obliczającej W celu obliczenia wartości splajnu należy w funkcji main (lub innym module wyższego poziomu) wywołać odpowiednią funkcję obliczeniową:

* 1 arytmetyka zmiennoprzecinkowa:

Obraz zawierający Czcionka, tekst, zrzut ekranu, Grafika

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

* 2 i 3 arytmetyka przedziałowa:

Obraz zawierający tekst, Czcionka, zrzut ekranu

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

1. Dane wejściowe  
   Plik wejściowy powinien zawierać kolejno:

* tryb (Integer): 1, 2 lub 3,
* n (Integer): liczba węzłów,
* n+1 wartości x i wartości y (od 0 do n):
  + tryb 1: liczb w formacie dziesiętnym,
  + tryb 2: pojedynczych wartości przedziałów dla x i y,
  + tryb 3: par granic przedziałów (dolna, górna) dla x i y,
* wartość xi (zgodnie z formatem punktu powyżej).

1. Wyniki  
   W pliku wyjściowym uzyskuje się:

* a[i,j]: współczynniki globalne powszechnie zdefiniowane, gdzie i=0..3, j=0..n-2,
* width: szerokość przedziału (tylko w trybach 2 i 3),
* S(xi): wartość splajnu w punkcie xi (oraz – dla przedziałów – szerokość rezultatu).

1. Inne parametry  
   Program zwraca zmienna st która podaje status obliczeń:

* 0 w przypadku sukcesu obliczeń
* 1 w przypadku błędu i nie możliwości obliczenia wartości

1. Typy parametrów  
   Integer: tryb, n, st  
   \_\_float128: x, y, xi, współczynniki splajnu w trybie 1  
   Interval<long double>: x, y, xi, współczynniki splajnu w trybach 2 i 3
2. Identyfikatory nielokalne  
   vector<\_\_float128> – dynamiczna tablica wartości typu \_\_float128  
   vector – dynamiczna tablica obiektów typu Interval  
   Interval – struktura przedziału z polami lo, hi typu long double
3. Tekst programu  
   Dla trybu 1 arytmetyki zmiennoprzecinkowej:
4. class NaturalCubicSpline {
5. private:
6. vector<\_\_float128> x, y, h;
7. vector<SplineSegment> segments;
8. public:
9. NaturalCubicSpline(const vector<\_\_float128>& x\_in, const vector<\_\_float128>& y\_in) {
10. x = x\_in; y = y\_in;
11. int n = x.size();
12. h.resize(n - 1);
13. for (int i = 0; i < n - 1; i++) {
14. h[i] = x[i + 1] - x[i];
15. }
16. // Układ równań dla naturalnego splajnu
17. vector<\_\_float128> alpha(n, 0.0Q), l(n, 0.0Q), mu(n, 0.0Q), z(n, 0.0Q);
18. l[0] = 1.0Q; mu[0] = 0.0Q; z[0] = 0.0Q;
19. for (int i = 1; i < n - 1; i++) {
20. alpha[i] = 6.0Q \* ((y[i + 1] - y[i]) / h[i] - (y[i] - y[i - 1]) / h[i - 1]);
21. l[i] = 2.0Q \* (x[i + 1] - x[i - 1]) - h[i - 1] \* mu[i - 1];
22. mu[i] = h[i] / l[i];
23. z[i] = (alpha[i] - h[i - 1] \* z[i - 1]) / l[i];
24. }
25. l[n - 1] = 1.0Q; z[n - 1] = 0.0Q;
26. vector<\_\_float128> c(n, 0.0Q), b(n - 1, 0.0Q), d(n - 1, 0.0Q);
27. c[n - 1] = 0.0Q;
28. for (int j = n - 2; j >= 0; j--) {
29. c[j] = z[j] - mu[j] \* c[j + 1];
30. b[j] = (y[j + 1] - y[j]) / h[j] - h[j] \* (c[j + 1] + 2.0Q \* c[j]) / 6.0Q;
31. d[j] = (c[j + 1] - c[j]) / (6.0Q \* h[j]);
32. }
33. // Wypełniamy segmenty
34. segments.resize(n - 1);
35. for (int i = 0; i < n - 1; i++) {
36. segments[i].a = y[i];
37. segments[i].b = b[i];
38. segments[i].c = c[i]; // zachowujemy oryginalne c[i]
39. segments[i].d = d[i];
40. segments[i].x = x[i];
41. \_\_float128 xi = x[i];
42. segments[i].a0 = segments[i].a - segments[i].b \* xi + (segments[i].c \* xi \* xi) / 2.0Q - segments[i].d \* xi \* xi \* xi;
43. segments[i].a1 = segments[i].b - segments[i].c \* xi + 3.0Q \* segments[i].d \* xi \* xi;
44. segments[i].a2 = (segments[i].c) / 2.0Q - 3.0Q \* segments[i].d \* xi;
45. segments[i].a3 = segments[i].d;
46. }
47. }
49. // Obliczenie S(xi) przy użyciu postaci lokalnej
50. tuple<\_\_float128, \_\_float128, \_\_float128, \_\_float128, \_\_float128> evaluate(\_\_float128 xi) {
51. int n = segments.size();
52. if (n == 0) return {0.0Q,0.0Q,0.0Q,0.0Q,0.0Q};
53. int seg = 0;
54. if (xi < x[0])
55. seg = 0;
56. else if (xi >= x[x.size()-1])
57. seg = x.size()-2;
58. else {
59. for (int i = 0; i < x.size()-1; i++) {
60. if (xi >= x[i] && xi < x[i+1]) { seg = i; break; }
61. }
62. }
63. \_\_float128 dx = xi - segments[seg].x;
64. \_\_float128 value = segments[seg].a + segments[seg].b \* dx +
65. (segments[seg].c / 2.0Q) \* dx \* dx +
66. segments[seg].d \* dx \* dx \* dx;
67. return {value, segments[seg].a, segments[seg].b, (segments[seg].c/2.0Q), segments[seg].d};
68. }
70. // Wypisanie współczynników globalnych (macierz a[0..3, 0..(n-2)])
71. void printCoefficients(ofstream& outputFile) {
72. const int numCoeff = 4;
73. int numSegments = segments.size();
74. for (int coeff = 0; coeff < numCoeff; coeff++) {
75. for (int seg = 0; seg < numSegments; seg++) {
76. char buffer[128];
77. \_\_float128 value;
78. if (coeff == 0)       value = segments[seg].a0;
79. else if (coeff == 1)  value = segments[seg].a1;
80. else if (coeff == 2)  value = segments[seg].a2;
81. else                  value = segments[seg].a3;
82. quadmath\_snprintf(buffer, sizeof(buffer), "%.18Qe", value);
83. outputFile << "a[" << coeff << "," << seg << "] = " << buffer << "\n";
84. }
85. }
86. }
87. };

Dla trybu 2 i 3 arytmetyka przedziałowa:

class NaturalCubicSplineInterval {

private:

    vector<Interval> x, y, h;

    vector<IntervalSplineSegment> segments;

public:

    // Konstruktor przyjmujący wektory przedziałów dla x i y

    NaturalCubicSplineInterval(const vector<Interval>& x\_in, const vector<Interval>& y\_in) {

        x = x\_in; y = y\_in;

        int n = x.size();

        h.resize(n - 1);

        for (int i = 0; i < n - 1; i++) {

            h[i] = subInt(x[i+1], x[i]);

            // Sprawdzamy, czy h[i] zawiera zero

            if (h[i].lo <= 0 && h[i].hi >= 0) {

                throw std::invalid\_argument("Przedział h[i] zawiera zero, co uniemożliwia konstrukcję splajnu");

            }

        }

        // Przygotowanie układu równań – wszystkie zmienne jako przedziały

        vector<Interval> alpha(n, I(0.0Q)), l(n, I(0.0Q)), mu(n, I(0.0Q)), z(n, I(0.0Q));

        l[0] = I(1.0Q); mu[0] = I(0.0Q); z[0] = I(0.0Q);

        for (int i = 1; i < n - 1; i++) {

            // alpha[i] = 6 \* [ (y[i+1]-y[i])/h[i] - (y[i]-y[i-1])/h[i-1] ]

            Interval diff1 = divInt( subInt(y[i+1], y[i]), h[i] );

            Interval diff2 = divInt( subInt(y[i], y[i-1]), h[i-1] );

            alpha[i] = mul( I(6.0Q), subInt(diff1, diff2) );

            l[i] = subInt( mul(I(2.0Q), subInt(x[i+1], x[i-1]) ), mul( h[i-1], mu[i-1] ) );

            mu[i] = divInt( h[i], l[i] );

            z[i] = divInt( subInt(alpha[i], mul( h[i-1], z[i-1] ) ), l[i] );

        }

        l[n - 1] = I(1.0Q); z[n - 1] = I(0.0Q);

        vector<Interval> c(n, I(0.0Q)), b(n - 1, I(0.0Q)), d(n - 1, I(0.0Q));

        c[n - 1] = I(0.0Q);

        for (int j = n - 2; j >= 0; j--) {

            c[j] = subInt(z[j], mul( mu[j], c[j+1] ) );

            Interval term = add( c[j+1], mul( I(2.0Q), c[j] ) );

            b[j] = subInt( divInt( subInt(y[j+1], y[j]), h[j] ),

                           divInt( mul( h[j], term ), I(6.0Q) ) );

            d[j] = divInt( subInt(c[j+1], c[j]), mul( I(6.0Q), h[j] ) );

        }

        // Wypełnienie segmentów – postać lokalna:

        // S\_i(x) = y[i] + b[i]\*(x-x[i]) + (c[i]/2)\*(x-x[i])^2 + d[i]\*(x-x[i])^3.

        segments.resize(n - 1);

        for (int i = 0; i < n - 1; i++) {

            segments[i].a = y[i];

            segments[i].b = b[i];

            segments[i].c = c[i]; // przechowujemy c[i] bez dzielenia przez 2

            segments[i].d = d[i];

            segments[i].x = x[i];

            // Przekształcenie do postaci globalnej:

            // a₀ = a - b\*x + (c\*x²)/2 - d\*x³

            Interval temp1 = subInt( segments[i].a, mul( segments[i].b, x[i] ) );

            Interval temp2 = divInt( mul( segments[i].c, square(x[i]) ), I(2.0Q) );

            Interval temp3 = mul( segments[i].d, cube(x[i]) );

            segments[i].a0 = subInt( add( temp1, temp2 ), temp3 );

            // a₁ = b - c\*x + 3\*d\*x²

            segments[i].a1 = add( subInt( segments[i].b, mul( segments[i].c, x[i] ) ),

                                  mul( I(3.0Q), mul( segments[i].d, square(x[i]) ) ) );

            // a₂ = (c/2) - 3\*d\*x

            segments[i].a2 = subInt( divInt( segments[i].c, I(2.0Q) ),

                                    mul( I(3.0Q), mul( segments[i].d, x[i] ) ) );

            segments[i].a3 = segments[i].d;

        }

    }

    // Obliczenie S(xi) (postać lokalna) – xi jest przedziałem

    tuple<Interval, Interval, Interval, Interval, Interval> evaluate(const Interval &xi) {

        int n = segments.size();

        if(n == 0) return {I(0.0Q), I(0.0Q), I(0.0Q), I(0.0Q), I(0.0Q)};

        int seg = 0;

        // Wybór segmentu analogicznie do trybu 1 – operujemy na dolnych i górnych granicach

        if(xi.lo < x[0].lo)

            seg = 0;

        else if(xi.hi >= x[x.size()-1].hi)

            seg = x.size()-2;

        else {

            for (int i = 0; i < x.size()-1; i++) {

                if(xi.lo >= x[i].lo && xi.hi < x[i+1].hi) { seg = i; break; }

            }

        }

        Interval dx = subInt(xi, segments[seg].x);

        Interval term1 = segments[seg].a;

        Interval term2 = mul(segments[seg].b, dx);

        Interval term3 = mul( divInt( segments[seg].c, I(2.0Q) ), square(dx) );

        Interval term4 = mul( segments[seg].d, cube(dx) );

        Interval value = add( add(term1, term2), add(term3, term4) );

        return {value, segments[seg].a, segments[seg].b, divInt(segments[seg].c, I(2.0Q)), segments[seg].d};

    }

    // Wypisanie współczynników globalnych – przedziały wypisywane w formacie "[lo, hi]"

    void printCoefficients(ofstream &outputFile) {

        const int numCoeff = 4;

        int numSegments = segments.size();

        for (int coeff = 0; coeff < numCoeff; coeff++) {

            for (int seg = 0; seg < numSegments; seg++) {

                Interval val;

                if (coeff == 0)       val = segments[seg].a0;

                else if (coeff == 1)  val = segments[seg].a1;

                else if (coeff == 2)  val = segments[seg].a2;

                else                  val = segments[seg].a3;

                // Wypisz przedział

                outputFile << "a[" << coeff << "," << seg << "] = ";

                IEndsToString(val, outputFile);

                outputFile << "\n";

                // Oblicz i wypisz szerokość w formacie X.Xe+X

                \_\_float128 width = IntWidth(val);

                char widthBuffer[128];

                quadmath\_snprintf(widthBuffer, sizeof(widthBuffer), "%.1Qe", width);

                outputFile << "width = " << widthBuffer << "\n\n";

            }

        }

    }

};

10. Przykłady:

* Dane liczbowe

Dane wejściowe:

tryb=1,n=6

x[0,6] =17 20 23 24 25 27 27.7

y[0,6] =4.5 7.0 6.1 5.6 5.8 5.2 4.1

XX = 23.5

Wyniki:

Wyniki:

a[0,0] = 1.344032189718982920e+02

a[0,1] = -3.109949430852148926e+02

a[0,2] = -2.478917395863841205e+03

a[0,3] = 3.521704513117991063e+03

a[0,4] = 1.315837790395570966e+03

a[0,5] = -6.630634615732411608e+03

a[1,0] = -2.513556622084497225e+01

a[1,1] = 4.167415808772200545e+01

a[1,2] = 3.244466519284123940e+02

a[1,3] = -4.256310866943166394e+02

a[1,4] = -1.609270799676262278e+02

a[1,5] = 7.220142984910385026e+02

a[2,0] = 1.543605917556053129e+00

a[2,1] = -1.796880297872295756e+00

a[2,2] = -1.409133655181535613e+01

a[2,3] = 1.716190255746502027e+01

a[2,4] = 6.573742288397403800e+00

a[2,5] = -2.612779024710869732e+01

a[3,0] = -3.026678269717751234e-02

a[3,1] = 2.540798755996163575e-02

a[3,2] = 2.035885129794262788e-01

a[3,3] = -2.304842524272456155e-01

a[3,4] = -8.930878217301072932e-02

a[3,5] = 3.144138417221263216e-01

S(2.350000000000000000e+01) = 5.784586596691450868e+00

St: 0

* Dane przedziałowe

Dane:

tryb=3 n=6

x[0,6]=

y[0,6]=

XX=

Wyniki:

a[0,0] = [2.744309282776075298e+01, 2.412608144899663103e+02]

width = 2.1e+02

a[0,1] = [-8.369121168994171318e+02, 3.542999840692136805e+02]

width = 1.2e+03

a[0,2] = [-1.087649908876649961e+04, 2.073593546558964253e+03]

width = 1.3e+04

a[0,3] = [-4.489656869590757846e+03, 1.510682129251849337e+04]

width = 2.0e+04

a[0,4] = [-2.961245591023110951e+03, 7.407197327284148524e+03]

width = 1.0e+04

a[0,5] = [-2.435699068934193160e+04, -1.316244601862284262e+02]

width = 2.4e+04

a[1,0] = [-4.310388834178538099e+01, -6.966215323688287248e+00]

width = 3.6e+01

a[1,1] = [-5.271083079793548336e+01, 1.151061961068889348e+02]

width = 1.7e+02

a[1,2] = [-2.592525803594414615e+02, 1.401815424749951841e+03]

width = 1.7e+03

a[1,3] = [-1.838534317287986728e+03, 5.548903461519244498e+02]

width = 2.4e+03

a[1,4] = [-8.658681921847170016e+02, 3.289100888009425706e+02]

width = 1.2e+03

a[1,5] = [1.980834808344265609e+01, 2.649890940952165036e+03]

width = 2.6e+03

a[2,0] = [4.957987608743753903e-01, 2.567851391427561761e+00]

width = 2.1e+00

a[2,1] = [-5.246892953762396782e+00, 2.691492731069075723e+00]

width = 7.9e+00

a[2,2] = [-6.019487058723244330e+01, 1.089711798951555325e+01]

width = 7.1e+01

a[2,3] = [-2.286915292941736951e+01, 7.463839275262402336e+01]

width = 9.8e+01

a[2,4] = [-1.217249648873084346e+01, 3.380401956479144888e+01]

width = 4.6e+01

a[2,5] = [-9.604449487004163986e+01, -8.065879568574389699e-01]

width = 9.5e+01

a[3,0] = [-5.005558267889983939e-02, -9.779068261821999809e-03]

width = 4.0e-02

a[3,1] = [-4.619190358843394562e-02, 8.004023315521787946e-02]

width = 1.3e-01

a[3,2] = [-1.536110667462407173e-01, 8.616591564469856703e-01]

width = 1.0e+00

a[3,3] = [-1.010405281087241965e+00, 3.146937404319151220e-01]

width = 1.3e+00

a[3,4] = [-4.404455522436222944e-01, 1.505128120071433201e-01]

width = 5.9e-01

a[3,5] = [9.671318427547229854e-03, 1.159957667512580192e+00]

width = 1.2e+00

S([2.340000000000000000e+01, 2.360000000000000000e+01]) = [4.513531535587234369e+00, 6.746527636410563223e+00]

width = 2.2e+00

St: 0

* Dane z błędem

St: 1