

Учбова практика
Побудова інтерполяційного сплайну

Виконав:
студент 4-го курсу
спеціальність математика
Чаповський Євгеній

Постановка задачі

Необхідно побудувати інтерполяційний сплайн $S(x, u)$ другого степеня дефекту 1, з крайовими умовами типу II, використовуючи метод $2m$ для пошуку системи лінійних рівнянь та метод монотонної лівої прогонки для розв'язання цієї системи.

Теоретичні відомості

Інтерполяційний сплайн другого степеня дефекту 1 — це така функція $S(x, u) \in C^1([a, b])$, що для інтерполяційної сітки X та функції u виконується:

$$\forall x \in [X_i, X_{i+1}) : S(x, u) = a_2^i(x - X_i)^2 - a_1^i(x - X_i) + a_0^i; \forall i : S(X_i, u) = u(X_i).$$

Попередні умови породжують обмеження на коефіцієнти, а саме: $2n$ обмежень впливає з необхідності рівності сплайну та функції у вузлах сітки (по два обмеження на кожний відрізок $[x_i, x_{i+1})$) та $n - 1$ обмеження через неперервність похідної (по одному в кожній внутрішній точці сітки). З двома крайовими умовами отримуємо $3n + 1$ обмеження, але тільки $3n$ змінних, тому таку задачу неможливо розв'язати в загальному випадку. Вихід з цієї ситуації — використання нової сплайнової сітки $\{x_i\} \subset [a, b]$, точки якої зазвичай кладуть посередині відрізків інтерполяційної сітки:

$$x_i = \frac{X_{i-1} + X_i}{2}, i = \overline{2, n}; \quad x_1 = X_1, x_{n+1} = X_{n+1}.$$

Тоді

$$\forall x \in [x_i, x_{i+1}) : S(x, u) = a_2^i(x - X_i)^2 + a_1^i(x - X_i) + a_0^i; \forall i : S(X_i, u) = u(X_i).$$

Тоді кількість змінних і кількість умов співпадає.

Для побудови системи рівнянь для коефіцієнтів використаємо перші похідні $a_1^i = m_i = S'(X_i, u)$. Використовуючи рівність поділених різниць сплайну та

інтерпольованої функції в точках X отримуємо обмеження на m_i :

$$h_i m_{i-1} + 3(h_{i-1} + h_i)m_i + h_{i-1}m_{i+1} = 4(h_{i-1}u(X_i; X_{i+1}) + h_i u(X_{i-1}; X_i))$$

Де $h_i = X_{i+1} - X_i$. Тоді, з урахуванням крайових умов, отримуємо систему рівнянь:

$$\left\{ \begin{array}{l} 3m_1 + m_2 = 4u(X_1, X_2) - h_1 A; \\ h_i m_{i-1} + 3(h_{i-1} + h_i)m_i + h_{i-1}m_{i+1} = 4(h_{i-1}u(X_i; X_{i+1}) + h_i u(X_{i-1}; X_i)), \\ i = \overline{2, n-1}; \\ m_{n-1} + 3m_n = 4u(X_{n-1}, X_n) + h_{n-1}B; \end{array} \right.$$

Для якої можна записати матрицю

$$\left(\begin{array}{cccc|c} 3 & 1 & 0 & \dots & 4u(X_1, X_2) - h_1 A \\ h_2 & 3(h_1 + h_2) & h_1 & \dots & 4(h_1 u(X_2; X_3) + h_2 u(X_1; X_2)) \\ \dots & \dots & \dots & \dots & \dots \\ \dots & h_{n-1} & 3(h_{n-2} + h_{n-1}) & h_{n-2} & 4(h_{n-2} u(X_{n-1}; X_n) + h_{n-1} u(X_{n-2}; X_{n-1})) \\ \dots & 0 & 1 & 3 & 4u(X_{n-1}, X_n) + h_{n-1} B \end{array} \right)$$

З теорії побудови сплайну відомо, що отримана система матиме властивість діагонального домінування, тому метод монотонної лівої прогонки гарантовано знайде розв'язок.

Для обрахування саме коефіцієнтів сплайну використовуються наступні формули:

$$\left\{ \begin{array}{l} a_0^i = u_i; \\ a_1^i = m_i; \\ a_2^1 = h_1^{-1}(2u(X_1; X_2) - 0.5(3m_1 + m_2)); \\ a_2^i = h_{i-1}^{-1}(0.5(m_{i-1} + 3m_i) - 2u(X_{i-1}; X_i)), \quad i = \overline{2, n}. \end{array} \right.$$

Практична реалізація

В практичній реалізації з метою упорядкування коду створено декілька функцій та трохи змінено їх роль у програмі. Обрахунок сплайну в точках сітки T відбувається безпосередньо в головній (перевіряючій) частині та ця сітка не передається в функцію що будує сплайн. Ця функція в реалізації має назву *CreateSpline*, вона повертає коефіцієнти побудованого сплайну (як матрицю $3 \times n$) та функцію, що обчислює сплайн та його похідні.

Функція, що здійснює перевірку правильності побудови сплайну: побудову графіків та розрахунок сіткової норми.

main.m

```
1 function [] = main(func, points, plotPoints, condition)
2     if ~exist('func')
3         func = @(t)(sin(2 * t) * sign(sin(t)));
4     end;
5     if ~exist('points')
6         points = 0 : 0.1 : 10;
7     end;
8     if ~exist('plotPoints')
9         plotPoints = 0 : 0.01 : 10;
10    end;
11    if ~exist('condition')
12        condition = [0, 0];
13    end;
14
15    [ interpolationSpline, splineFunc] = CreateSpline(points, func, condition);
16    splineVal = @(t)(splineFunc(0, t));
17    splineDerivative = @(t)(splineFunc(1, t));
18    splineSecondDerivative = @(t)(splineFunc(2, t));
19
20    figure('units','normalized','outerposition',[0 0 1 1], 'paperorientation',
21           'landscape');
22    if strcmp(class(func), 'function_handle')
23        plot(plotPoints, arrayfun(func, plotPoints), 'k--', plotPoints,
24             arrayfun(splineVal, plotPoints), 'k', points, arrayfun(func, points), 'kx');
25        legend('interpolated _function', 'interpolation _spline', 'pivot_points',
26              'location', 'southoutside');
27        title(sprintf('Maximal_deviation:_%e', max(abs(arrayfun(func, plotPoints) -
28              arrayfun(splineVal, plotPoints)))));
29    else
30        plot(plotPoints, arrayfun(splineVal, plotPoints), 'k', points, func, 'kx');
31        legend('interpolation _spline', 'pivot_points', 'location', 'southoutside');
32    end;
33    grid minor;
34    print -dpdf ./ result .pdf;
```

```

31 figure('units','normalized','outerposition',[0 0 1 1], 'paperorientation',
    'landscape');
32 plot(plotPoints, arrayfun(splineDerivative, plotPoints), 'k--', plotPoints,
    arrayfun(splineSecondDerivative, plotPoints), 'k');
33 legend('spline_first_derivative', 'spline_second_derivative', 'location',
    'southoutside');
34 grid minor;
35 print -dpdf -append ./result .pdf;
36 end;

```

Функція, що здійснює побудову сплайна.

CreateSpline.m

```

1 function [ interpolationSpline, splineFunction ] = CreateSpline(points, func, condition)
2     if strcmp(class(func), 'function_handle')
3         values = arrayfun(func, points);
4     elseif length(func) == length(points)
5         values = func;
6     else
7         error('Unknown_format_of_input_argument_func. ');
8     end;
9     if isrow(points)
10         points = points';
11     end;
12     if isrow(values)
13         values = values';
14     end;
15
16     [ matrix, splinePoints ] = CreateSEMatrix(points, values, condition);
17     solution = SolveSE(matrix);
18     interpolationSpline = FormSpline(points, values, solution);
19     splineFunction = @(derivative, t)( EvaluateSpline(points, splinePoints,
        interpolationSpline, derivative, t));
20 end;
21
22 function result = EvaluateSpline(points, splinePoints, interpolationSpline, derivative,
    t)
23     [row, relativeValue] = SelectRow(points, splinePoints, interpolationSpline, t);
24     coefficients = EvaluateCoefficients(length(row), derivative);
25     powers = relativeValue .^ (length(row) - derivative - 1 : -1 : 0);
26     result = sum(row(1 : length(powers)) .* powers .* coefficients(1 : length(powers)));
27 end;
28
29 function coefficients = EvaluateCoefficients(rowLength, derivative)
30     if derivative == 0
31         coefficients = ones(1, rowLength);
32         return;
33     end;
34     coefficients = prod((ones(derivative, 1) * (rowLength - 1 : -1 : 0)) - ((0 :
        derivative - 1)' * ones(1, rowLength)), 1);

```

```

35 end;
36
37 function [row, relativeValue] = SelectRow(points, splinePoints, interpolationSpline, t)
38     index = max([0; find((t - splinePoints) >= 0)]) + 1;
39     row = interpolationSpline(index, :);
40     relativeValue = t - points(index);
41 end;

```

Побудова матриці за допомогою перших похідних m_i .

CreateSEMatrix.m

```

1 function [matrix, splinePoints] = CreateSEMatrix(points, values, condition)
2     pointsCount = length(points);
3     segments = points(2 : end) - points(1 : end - 1);
4     splinePoints = points(2 : end) - segments / 2;
5     deltas = (values(2 : end) - values(1 : end - 1)) ./ segments(1 : end);
6     matrix = [diag(segments(2 : end)), zeros(pointsCount - 2, 2)] + [zeros(pointsCount -
7         2, 2), diag(segments(1 : end - 1))] + ...
8     3 * [zeros(pointsCount - 2, 1), diag(segments(1 : end - 1)) + diag(segments(2 :
9         end)), zeros(pointsCount - 2, 1)];
10    matrix = [3, 1, zeros(1, pointsCount - 2); matrix; zeros(1, pointsCount - 2), 1, 3];
11    rightSide = [4 * deltas(1) - segments(1) * condition(1); ...
12    4 * (deltas(2 : end) .* segments(1 : end - 1) + deltas(1 : end - 1) .* segments(2
13        : end)); ...
14    4 * deltas(end) + segments(end) * condition(2)];
15    matrix = [matrix, rightSide];
16 end;

```

Розв'язання системи лінійних рівнянь за допомогою методу монотонної лівої прогонки.

SolveSE.m

```

1 function solution = SolveSE(matrix)
2     [rows, cols] = size(matrix);
3     core = matrix(:, 1 : rows);
4     mask = diag(ones(1, rows)) + diag(ones(1, rows - 1), 1) + diag(ones(1, rows - 1),
5         -1);
6     if max(abs(core - core .* mask)) < 1e-10
7         %for used formulae see Popov's book
8         rightSide = matrix(:, rows + 1 : end);
9         gammas = deltas = zeros(rows + 1, cols - rows);
10        for i = rows : -1 : 2
11            gammas(i, :) = -matrix(i, i - 1) ./ (matrix(i, i) + gammas(i + 1, :) *
12                matrix(i, i + 1));
13            deltas(i, :) = (rightSide(i, :) - matrix(i, i + 1) .* deltas(i + 1, :)) ./
14                (matrix(i, i) + gammas(i + 1, :) * matrix(i, i + 1));
15        end;
16    end;

```

```

13     solution = zeros(rows, cols - rows);
14     solution (1, :) = (rightSide (1, :) - matrix(1, 2) * deltas (2, :)) ./ (matrix(1,
15         1) + gammas(2, :) * matrix(1, 2));
16     for i = 2 : rows
17         solution (i, :) = gammas(i, :) .* solution (i - 1, :) + deltas (i, :);
18     end;
19 else
20     error('Matrix is not tridiagonal ');
21 end;

```

Формування коефіцієнтів сплайну.

FormSpline.m

```

1 function interpolationSpline = FormSpline(points, values, solution)
2     pointsCount = length(points);
3     segments = points (2 : end) - points(1 : end - 1);
4     deltas = (values(2 : end) - values(1 : end - 1)) ./ segments(1 : end);
5
6     interpolationSpline = [zeros(pointsCount, 1), solution, values];
7     interpolationSpline (:, 1) = [(2 * deltas (1) - 0.5 * (3 * solution (1) + solution (2)))
8         / segments(1) ;...
9     (0.5 * (solution (1 : end - 1) + 3 * solution (2 : end)) - 2 * deltas ) ./ segments];
10 end;

```
