КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Учбова практика

Побудова інтерполяційного сплайну

Виконав:
студент 4-го курсу
спеціальність математика
Чаповський Євгеній

2016

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 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 випливає 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:

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

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:

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

Для якої можна записати матрицю

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

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:

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

Практична реалізація

В практичній реалізації з метою упорядкування коду створено декілька функцій та трохи змінено їх роль у программі. Обрахунок сплайну в точках сітки T відбувається безпосередньо в головній (перевіряючій) частині та ця сітка не передається в функцію що будує сплайн. Дійсно, для побудови сплайну не має бути важливо в яких точках він буде потім обраховуватись. Ця функція, що могла би називатися spl_22, в реалізації має назву $CreateSpline$, вона повертає коефіцієнти побудованого сплайну (як матрицю $3 \times n$) та функцію, що обраховує сплайн та його похідні.

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

main.m

```
function [] = main(func, points, plotPoints, condition)
    if ~exist('func')
        func = @(t)(sin(t^2));
    end;
    if ~exist('points')
        points = sqrt(0 : 0.05 : 1) * 5;
    end;
    if ~exist('plotPoints')
        plotPoints = 0 : 0.001 : 5;
    end;
    if ~exist('condition')
        condition = [0, 0];
    end;

    [interpolationSpline, splineFunc] = CreateSpline(points, func, condition);
    splineVal = @(t)(splineFunc(0, t));
    splineDerivative = @(t)(splineFunc(1, t));
    splineSecondDerivative = @(t)(splineFunc(2, t));

    figure('units','normalized','outerposition',[0 0 1 1], 'paperorientation',
        'landscape');
    if strcmp(class(func), 'function_handle')
        plot(plotPoints, arrayfun(func, plotPoints), 'k--', plotPoints,
            arrayfun(splineVal, plotPoints), 'k', points, arrayfun(func, points), 'kx');
        legend('interpolated function', 'interpolation spline', 'pivot points',
            'location', 'southoutside');
    else
        plot(plotPoints, arrayfun(splineVal, plotPoints), 'k', points, arrayfun(func,
            points), 'kx');
        legend('interpolation spline', 'pivot points', 'location', 'southoutside');
```

```
    end;
    title(sprintf('Maximal deviation: %e', max(abs(arrayfun(func, plotPoints) - arrayfun(splineVal, plotPoints)))));
    grid minor;
    print -dpdf ./result.pdf;
    figure('units','normalized','outerposition',[0 0 1 1], 'paperorientation', 'landscape');
    plot(plotPoints, arrayfun(splineDerivative, plotPoints), 'k--', plotPoints, arrayfun(splineSecondDerivative, plotPoints), 'k');
    legend('spline first derivative', 'spline second derivative', 'location', 'southoutside');
    grid minor;
    print -dpdf -append ./result.pdf;
end;
```

Функція, що здійснює побудову сплайна.

CreateSpline.m

```
%spl_22
function [interpolationSpline, splineFunction] = CreateSpline(points, func, condition)
    if strcmp(class(func), 'function_handle')
        values = arrayfun(func, points);
    elseif length(func) == length(points)
        values = func;
    else
        error('Unknown format of input argument func.');
    end;
    if isrow(points)
        points = points';
    end;
    if isrow(values)
        values = values';
    end;

    [matrix, splinePoints] = CreateSEMatrix(points, values, condition);
    solution = SolveSE(matrix);
    interpolationSpline = FormSpline(points, values, solution);
    splineFunction = @(derivative, t)(EvaluateSpline(points, splinePoints, interpolationSpline, derivative, t));
end;

function result = EvaluateSpline(points, splinePoints, interpolationSpline, derivative, t)
    [row, relativeValue] = SelectRow(points, splinePoints, interpolationSpline, t);
    coefficients = EvaluateCoefficients(length(row), derivative);
    powers = relativeValue .^ (length(row) - derivative - 1 : -1 : 0);
    result = sum(row(1 : length(powers)) .* powers .* coefficients(1 : length(powers)));
end;

function coefficients = EvaluateCoefficients(rowLength, derivative)
```

```
    if derivative == 0
        coefficients = ones(1, rowLength);
        return;
    end;
    coefficients = prod((ones(derivative, 1) * (rowLength - 1 : -1 : 0)) - ((0 : derivative - 1)' * ones(1, rowLength)), 1);
end;

function [row, relativeValue] = SelectRow(points, splinePoints, interpolationSpline, t)
    index = max([0; find((t - splinePoints) >= 0)]) + 1;
    row = interpolationSpline(index, :);
    relativeValue = t - points(index);
end;
```

Побудова матриці за допомогою перших похідних m_i.

CreateSEMatrix.m

```
function [matrix, splinePoints] = CreateSEMatrix(points, values, condition)
    pointsCount = length(points);
    segments = points(2 : end) - points(1 : end - 1);
    splinePoints = points(2 : end) - segments / 2;
    deltas = (values(2 : end) - values(1 : end - 1)) ./ segments(1 : end);
    matrix = [diag(segments(2 : end)), zeros(pointsCount - 2, 2)] + [zeros(pointsCount - 2, 2), diag(segments(1 : end - 1))] +...
    3 * [zeros(pointsCount - 2, 1), diag(segments(1: end - 1)) + diag(segments(2 : end)), zeros(pointsCount - 2, 1)];
    matrix = [3, 1, zeros(1, pointsCount - 2); matrix; zeros(1, pointsCount - 2), 1, 3];
    rightSide = [4 * deltas(1) - segments(1) * condition(1) ;...
    4 * (deltas(2 : end) .* segments(1 : end - 1) + deltas(1 : end - 1) .* segments(2 : end)) ;...
    4 * deltas(end) + segments(end) * condition(2)];
    matrix = [matrix, rightSide];
end;
```

Розв'язання системи лінійних рівнянь за допомогою методу квадратного кореня.

SolveSE.m

```
function solution = SolveSE(matrix)
    [rows, cols] = size(matrix);
    core = matrix(:, 1 : rows);
    mask = diag(ones(1, rows)) + diag(ones(1, rows - 1), 1) + diag(ones(1, rows - 1), -1);
    if max(abs(core - core .* mask)) < 1e-10
        %for used formulae see Popov's book
        rightSide = matrix(:, rows + 1 : end);
        gammas = deltas = zeros(rows + 1, cols - rows);
```

```
        for i = rows : -1 : 2
            gammas(i, :) = -matrix(i, i - 1) ./ (matrix(i, i) + gammas(i + 1, :) * matrix(i, i + 1));
            deltas(i, :) = (rightSide(i, :) - matrix(i, i + 1) .* deltas(i + 1, :)) ./ (matrix(i, i) + gammas(i + 1, :) * matrix(i, i + 1));
        end;
        solution = zeros(rows, cols - rows);
        solution(1, :) = (rightSide(1, :) - matrix(1, 2) * deltas(2, :)) ./ (matrix(1, 1) + gammas(2, :) * matrix(1, 2));
        for i = 2 : rows
            solution(i, :) = gammas(i, :) .* solution(i - 1, :) + deltas(i, :);
        end;
    else
        error('Matrix is not tridiagonal');
    end;
end;
```

Формування коефіцієнтів сплайну.

FormSpline.m

```
function interpolationSpline = FormSpline(points, values, solution)
    pointsCount = length(points);
    segments = points(2 : end) - points(1 : end - 1);
    deltas = (values(2 : end) - values(1 : end - 1)) ./ segments(1 : end);

    interpolationSpline = [zeros(pointsCount, 1), solution, values];
    interpolationSpline(:, 1) = [(2 * deltas(1) - 0.5 * (3 * solution(1) + solution(2))) / segments(1);...
    (0.5 * (solution(1 : end - 1) + 3 * solution(2 : end)) - 2 * deltas) ./ segments];
end;
```