КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Учбова практика

Побудова інтерполяційного сплайну

Виконав:

студент 4-го курсу

спеціальність математика

Шатохін Михайло

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 квадратного кореня 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 другі похідні $2a_i = M_i = S''(X_i, u)$. Використовуючи рівність поділених різниць сплайну та інтерпольованої функції в точках X отримуємо обмеження на M_i:

$$h_{i-1}M_{i-1} + 3(h_{i-1} + h_i)M_i + h_iM_i = 8u(X_{i-1}; X_i; X_{i+1})(h_{i-1} + h_i);$$
$$h_{i-1}M_{i-1} + 3(h_{i-1} + h_i)M_i + h_iM_i = 8(u(X_i; X_{i+1}) - u(X_{i-1}; X_i)).$$

Де $h_i = X_{i+1} - X_i$. Тоді отримуємо систему рівнянь:

$M_1 = A;$

$h_{i-1}M_{i-1} + 3(h_{i-1} + h_i)M_i + h_iM_i = 8(u(X_i; X_{i+1}) - u(X_{i-1}; X_i)), i = \overline{2, n-1};$

$M_n = B;$

Для якої можна записати матрицю

$$\left(\begin{array}{ccccc|c} 1 & 0 & 0 & \dots & 0 & A \\ h_1 & 3(h_1 + h_2) & h_2 & \dots & 0 & 8(u(X_2; X_3) - u(X_1; X_2)) \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & 0 & h_{n-2} & 3(h_{n-2} + h_{n-1}) & h_{n-1} & 8(u(X_{n-1}; X_n) - u(X_{n-2}; X_{n-1}) \\ 0 & \dots & 0 & 0 & 1 & B \end{array}\right)$$

В умові вимагається розв'язання системи лінійний рівнянь методом квадратного кореня, який в свою чергу вимагає ермітовості матриці, тому цю систему перетворюємо в симетричну відніманням від другого та передостаннього рядка відповідно першого та останнього, помноженних на відповідні коефіцієнти.

Для обрахування саме коефіцієнтів сплайну використовуються наступні формули:

$$a_i = \frac{M_i}{2};$$
$$c_i = u_i;$$
$$b_1 = u(X_1, X_2) - \frac{1}{8}h_1(3M_1 + M_2);$$
$$b_i = u(X_{i-1}; X_i) + \frac{1}{8}h_1(M_{i-1} + 3M_i), i = \overline{2, n}.$$

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 других похідних $M_i = 2N_i$.

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(1 : end - 1)), zeros(pointsCount - 2, 2)] + [zeros(pointsCount - 2, 2), diag(segments(2 : end))] +...
    3 * [zeros(pointsCount - 2, 1), diag(segments(1: end - 1)) + diag(segments(2 : end)), zeros(pointsCount - 2, 1)];
    matrix = [1, zeros(1, pointsCount - 1); matrix; zeros(1, pointsCount - 1), 1];
    rightSide = [condition(1); 8 * (deltas(2 : end) - deltas(1 : end - 1)); condition(2)];
    matrix = [matrix, rightSide];

    matrix(2, :) -= matrix(1, :) * matrix(2, 1);
    matrix(end - 1, :) -= matrix(end, :) * matrix(end - 1, end - 1);
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);
    if max(abs(core - conj(core'))) < 1e-10
        %for used formulae see Popov's book
        D = zeros(rows, 1);
        S = zeros(rows);
        for i = 1 : rows
```

```
            D(i) = sign(core(i, i) - sum(D(1:i - 1) .* (S(1 : i - 1, i) .* conj(S(1 :
                i - 1, i)))));
            S(i, i) = sqrt(abs( core(i, i) - sum(D(1:i - 1) .* (S(1 : i - 1, i) .*
                conj(S(1 : i - 1, i)))) ));
            for j = i + 1 : rows
                S(i, j) = (core(i, j) - sum(D(1:i - 1) .* S(1 : i - 1, i) .* S(1 : i -
                    1, j))) / (conj(S(i, i)) * D(i));
            end;
        end;
        rightSide = matrix(:, rows + 1 : end);
        v = zeros(rows, cols - rows);
        for i = 1 : rows
            v(i, :) = ( rightSide(i, :) - sum(((conj(S(1 : i - 1, i)) .* D(1 : i - 1)) *
                ones(cols - rows, 1)) .* v(1 : i - 1, :)) ) / (S(i, i) * D(i));
        end;
        solution = zeros(rows, cols - rows);
        for i = rows : -1 : 1
            solution(i, :) = (v(i, :) - sum( (S(i, i + 1 : end) * ones(cols - rows, 1))
                .* solution(i + 1 : end, :)')) / S(i, i);
        end;
    else
        error('Matrix is not hermitian');
    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 = [solution / 2, zeros(pointsCount, 1), values];
    interpolationSpline(:, 2) = [deltas(1) - 0.125 * segments(1) * (3 * solution(1) +
        solution(2)) ;...
    deltas(1 : end) + 0.125 * segments(1 : end) .* (solution(1 : end - 1) + 3 *
        solution(2 : end))];
end;
```