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.

Теоретичні відомості

За умовою задачі необхідно побудувати інтерполяційний сплайн $S(x,u)$ другого степеня дефекту 1, тобто за даною сіткою $X = (a = x_1 < x_2 < \ldots < x_{n_1} = b)$ побудувати функцію з неперервною першою похідною таку, що

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

Якщо сітка має $n+1$ точку, то 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. Скоротимо кількість крайових умов до однієї — вимагатимемо тільки в лівому кінці проміжку інтерполювання:

$$S''(x,u) = A;$$

Для побудови системи рівнянь для коефіцієнтів використаємо другі похідні $M_i = S''(x_i+, u)$, де береться права границя через те, що друга похідна сплайна не обов'язково неперервна. Згадавши явний вигляд сплайна отримуємо $a_2^i = \frac{M_i}{2}$, але для скорочення коефіцієнтів покладемо $N_i = a_2^i = \frac{M_i}{2}$. Тепер запишемо умови в термінах

нових змінних:

$1.S(x_i, u) = u(x_i) = u_i \implies a_0^i = u_i;$

Покладемо $x_{i+1} - x_i = h_i, \Delta_i = u_{i+1} - u_i;$

$2.S(x_{i+1}-, u) = u_{i+1} \implies N_i h_i^2 + a_1^i h_i = u_{i+1} - u_i \implies a_1^i = \dfrac{\Delta_i}{h_i} - N_i h;$

$3.S'(x_{i+1}-, u) = S'(x_i+, u) \implies 2N_i h_i + a_1^i = a_1^{i+1} \implies$

$\dfrac{\Delta_i}{h_i} + N_i h_i = \dfrac{\Delta_{i+1}}{h_{i+1}} - N_{i+1} h_{i+1} \implies N_i h_i + N_{i+1} h_{i+1} = \dfrac{\Delta_{i+1}}{h_{i+1}} - \dfrac{\Delta_i}{h_i};$

$4.S''(a, u) = A \implies 2N_1 = A.$

Таким чином отримаємо систему лінійний рівнянь для N_i:

$$\left(\begin{array}{ccccc|c} 2 & 0 & 0 & \dots & 0 & A \\ h_1 & h_2 & 0 & \dots & 0 & \frac{\Delta_2}{h_2} - \frac{\Delta_1}{h_1} \\ 0 & h_2 & h_3 & \dots & 0 & \frac{\Delta_3}{h_3} - \frac{\Delta_2}{h_2} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & h_{n-1} & h_n & \frac{\Delta_n}{h_n} - \frac{\Delta_{n-1}}{h_{n-1}} \end{array}\right)$$

В умові вимагається розв'язання системи лінійний рівнянь методом квадратного кореня, який в свою чергу вимагає ермітовості матриці, тому цю систему перетворюємо в симетричну додаванням до кожного рядка наступного, помноженого на відповідний коефіцієнт.

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_21, в реалізації має назву $CreateSpline$, вона повертає коефіцієнти побудованого сплайну (як матрицю $3xn$) 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, leftCondition)
    if ~exist('func')
        func = @(t)(sin(t^2));
    end;
    if ~exist('points')
        points = sqrt(0 : 0.2 : 1) * 5;
    end;
    if ~exist('plotPoints')
        plotPoints = 0 : 0.01 : 5;
    end;
    if ~exist('leftCondition')
        leftCondition = 0;
    end;

    [interpolationSpline, splineFunc] = CreateSpline(points, func, leftCondition);
    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');
    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');
    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

```
function [interpolationSpline, splineFunction] = CreateSpline(points, func,
    leftCondition)
    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      matrix = CreateSEMatrix(points, values, leftCondition);
16      solution = SolveSE(matrix);
17      interpolationSpline = FormSpline(points, values, solution);
18      splineFunction = @(derivative, t)(EvaluateSpline(points, interpolationSpline,
            derivative, t));
19  end;
20
21  function result = EvaluateSpline(points, interpolationSpline, derivative, t)
22      [row, relativeValue] = SelectRow(points, interpolationSpline, t);
23      if row == 0
24          result = 0;
25          return;
26      end;
27      coefficients = EvaluateCoefficients(length(row), derivative);
28      powers = relativeValue .^ (length(row) - derivative - 1 : -1 : 0);
29      result = sum(row(1 : length(powers)) .* powers .* coefficients(1 : length(powers)));
30  end;
31
32  function coefficients = EvaluateCoefficients(rowLength, derivative)
33      if derivative == 0
34          coefficients = ones(1, rowLength);
35          return;
36      end;
37      coefficients = prod((ones(derivative, 1) * (rowLength - 1 : -1 : 0)) - ((0 :
            derivative - 1)' * ones(1, rowLength)), 1);
38  end;
39
40  function [row, relativeValue] = SelectRow(points, interpolationSpline, t)
41      if t < points(1)
42          row = 1;
43          relativeValue = 0;
44          return;
45      end;
46      if t >= points(end)
47          row = interpolationSpline(end, :);
48          relativeValue = t - points(end - 1);
49          return;
50      end;
51
52      points = t - points;
```

```
53      interpolationSpline = interpolationSpline(points >= 0, :);
54      row = interpolationSpline(end, :);
55      points = points(points >= 0);
56      relativeValue = points(end);
57  end;
```

Побудова матриці за допомогою других похідних $M_i = 2N_i$.

CreateSEMatrix.m

```
1   function matrix = CreateSEMatrix(points, values, leftCondition)
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       matrix = diag(segments) + diag(segments(2: end), 1);
6       matrix = matrix(1 : end - 1, :);
7       matrix = [2, zeros(1, pointsCount - 2); matrix];
8       rightSide = [leftCondition; deltas(2 : end) - deltas(1 : end - 1)];
9       matrix = [matrix, rightSide];
10      for i = 1 : pointsCount - 2
11          matrix(i, :) += matrix(i + 1, :) * matrix(i + 1, i) / matrix(i + 1, i + 1);
12      end;
13  end;
```

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

SolveSE.m

```
1   function solution = SolveSE(matrix)
2       [rows, cols] = size(matrix);
3       core = matrix(:, 1 : rows);
4       if max(abs(core - conj(core'))) < 1e-10
5           %for used formulae see Popov's book
6           D = zeros(rows, 1);
7           S = zeros(rows);
8           for i = 1 : rows
9               D(i) = sign(core(i, i) - sum(D(1 : i - 1) .* (S(1 : i - 1, i) .* conj(S(1 :
                    i - 1, i)))));
10              S(i, i) = sqrt(abs(core(i, i) - sum(D(1 : i - 1) .* (S(1 : i - 1, i) .*
                    conj(S(1 : i - 1, i))))));
11              for j = i + 1 : rows
12                  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));
13              end;
14          end;
15          rightSide = matrix(:, rows + 1 : end);
16          v = zeros(rows, cols - rows);
17          for i = 1 : rows
```

```
18              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));
19          end;
20          solution = zeros(rows, cols - rows);
21          for i = rows : -1 : 1
22              solution(i, :) = (v(i, :) - sum( (S(i, i + 1 : end) * ones(cols - rows, 1))
                    .* solution(i + 1 : end, :)')) / S(i, i);
23          end;
24      else
25          error('Matrix is not hermitian');
26      end;
27  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 = [solution, deltas - segments .* solution, values(1 : end - 1)];
7   end;
```