КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Лабораторна робота учбової практики

Виконав:
студент 4-го курсу
спеціальність математика
Шатохін Михайло

2016

1 Постановка задачі

Необхідно побудувати інтерполяційний сплайн $S(x, u)$ другого степеня дефекту 1, з крайовими умовами типу II.

2 Код

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 = 0 : 1 : 5;
    end;
    if ~exist('plotPoints')
        plotPoints = 0 : 0.01 : 5;
    end;
    if ~exist('leftCondition')
        leftCondition = 0;
    end;

    interpolationSpline = CreateSpline(points, func, leftCondition);
    splineVal = @(t)(EvaluateSpline(points, interpolationSpline, 0, t));
    splineDerivative = @(t)(EvaluateSpline(points, interpolationSpline, 1, t));
    splineSecondDerivative = @(t)(EvaluateSpline(points, interpolationSpline, 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');
    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');
    grid minor;
    print -dpdf -append ./result.pdf;
end;

function result = EvaluateSpline(points, interpolationSpline, derivative, t)
    [row, relativeValue] = SelectRow(points, interpolationSpline, t);
    if row == 0
```

```
36            result = 0;
37            return;
38        end;
39        coefficients = EvaluateCoefficients (length(row), derivative );
40        powers = relativeValue .^ (length(row) - derivative - 1 : -1 : 0);
41        result = sum(row(1 : length(powers)) .* powers .* coefficients (1 : length(powers)));
42    end;
43
44    function coefficients = EvaluateCoefficients (rowLength, derivative )
45        if derivative == 0
46            coefficients = ones(1, rowLength);
47            return;
48        end;
49        coefficients = prod((ones( derivative , 1) * (rowLength - 1 : -1 : 0)) - ((0 :
              derivative - 1)' * ones(1, rowLength)), 1);
50    end;
51
52    function [row, relativeValue ] = SelectRow(points, interpolationSpline , t)
53        if t < points(1)
54            row = 1;
55            relativeValue = 0;
56            return;
57        end;
58        if t >= points(end)
59            row = interpolationSpline (end, :);
60            relativeValue = t - points(end - 1);
61            return;
62        end;
63
64        points = t - points;
65        interpolationSpline = interpolationSpline ( points >= 0, :);
66        row = interpolationSpline (end, :);
67        points = points( points >= 0);
68        relativeValue = points(end);
69    end;
```

Функція, що здійснює побудову сплайна.

CreateSpline.m

```
1   function [ interpolationSpline , values] = CreateSpline( points , func, leftCondition )
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      matrix = CreateSEMatrix(points, values, leftCondition);
16      solution = SolveSE(matrix);
17      interpolationSpline = FormSpline(points, values, solution);
18  end;
```

Побудова матриці за допомогою других похідних M_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 = [1, 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));
```

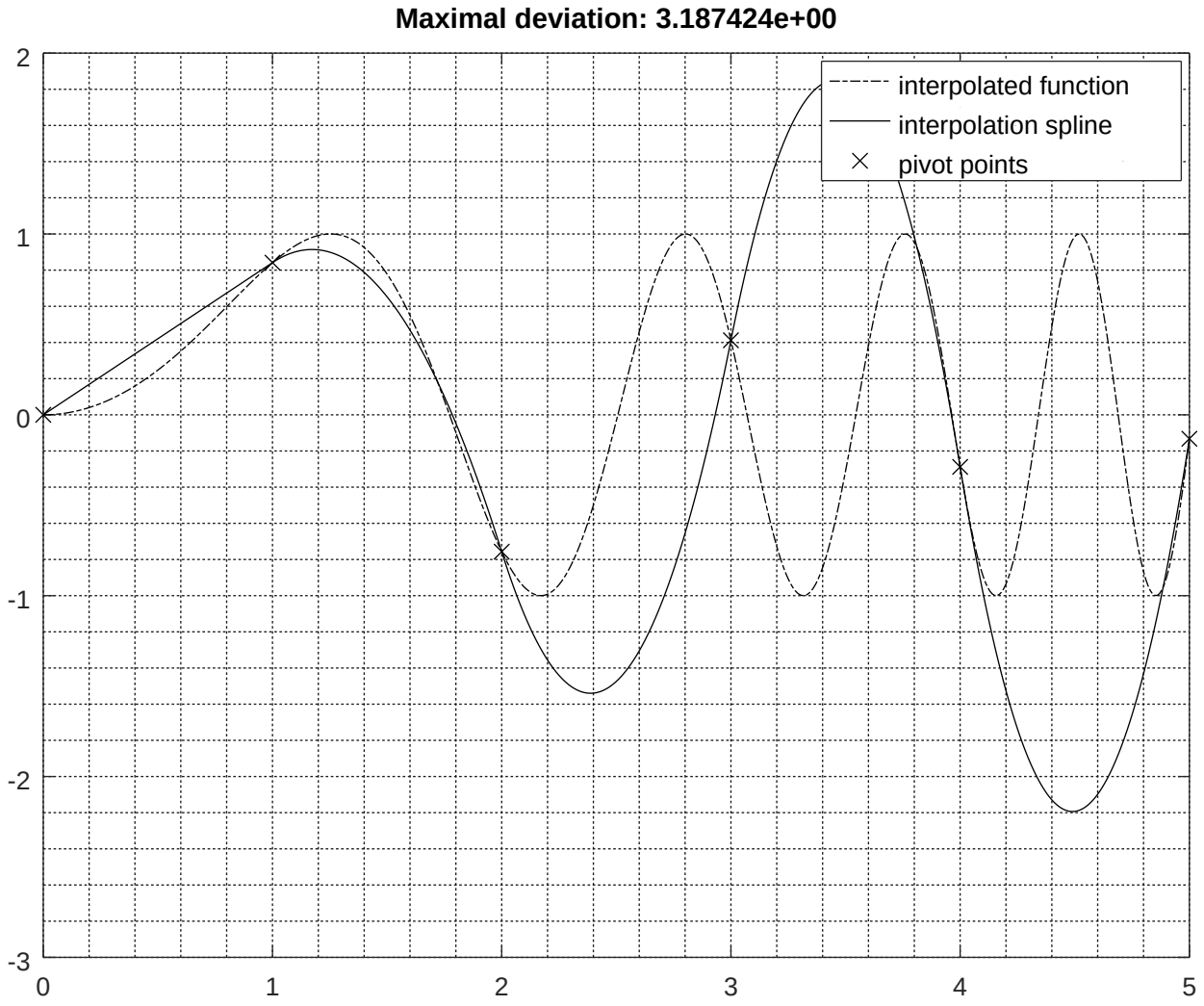
```
19            end;
20            solution = zeros(rows, cols - rows);
21            for i = rows : -1 : 1
22                solution(i, :) = (v(i, :) - sum((S(i, i + 1 : end) * ones(cols - rows, 1))
                      .* solution(i + 1 : end, :)')) / S(i, i);
23            end;
24        else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

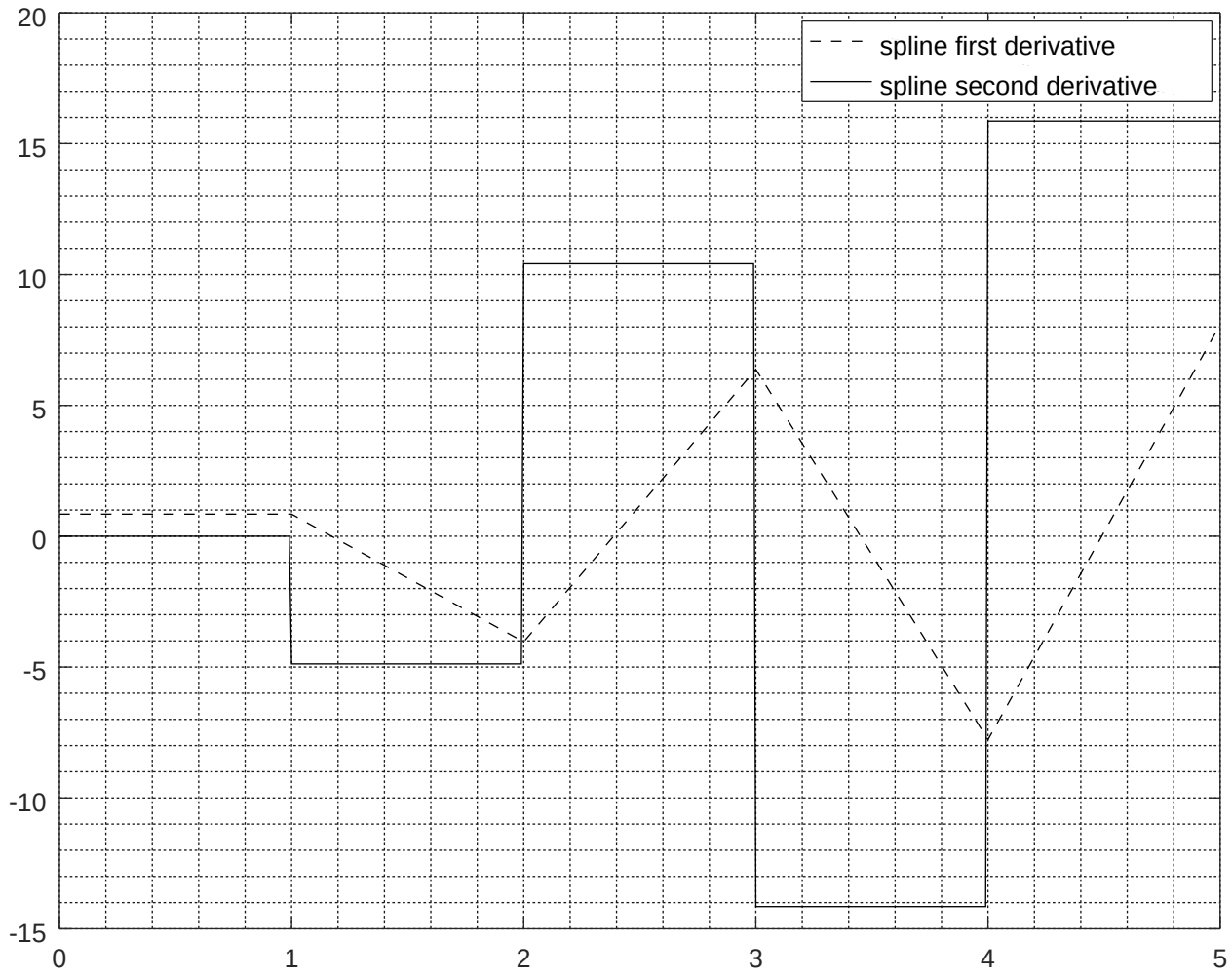
Формування коефіцієнтів сплайну.

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;
```

Нижче наведений результат роботи програми:


Maximal deviation: 3.187424e+00
interpolated function
interpolation spline
pivot points
2
1
0
-1
-2
-3
0
1
2
3
4
5


spline first derivative
spline second derivative
20
15
10
5
0
-5
-10
-15
0
1
2
3
4
5