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 = sqrt(0 : 0.2 : 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', '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;

function result = EvaluateSpline(points, interpolationSpline, derivative, t)
```

```
    [row, relativeValue] = SelectRow(points, interpolationSpline, t);
    if row == 0
        result = 0;
        return;
    end;
    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, interpolationSpline, t)
    if t < points(1)
        row = 1;
        relativeValue = 0;
        return;
    end;
    if t >= points(end)
        row = interpolationSpline(end, :);
        relativeValue = t - points(end - 1);
        return;
    end;

    points = t - points;
    interpolationSpline = interpolationSpline(points >= 0, :);
    row = interpolationSpline(end, :);
    points = points(points >= 0);
    relativeValue = points(end);
end;
```

Функція, що здійснює побудову сплайна.

CreateSpline.m

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

Побудова матриці за допомогою других похідних M_i.

CreateSEMatrix.m

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

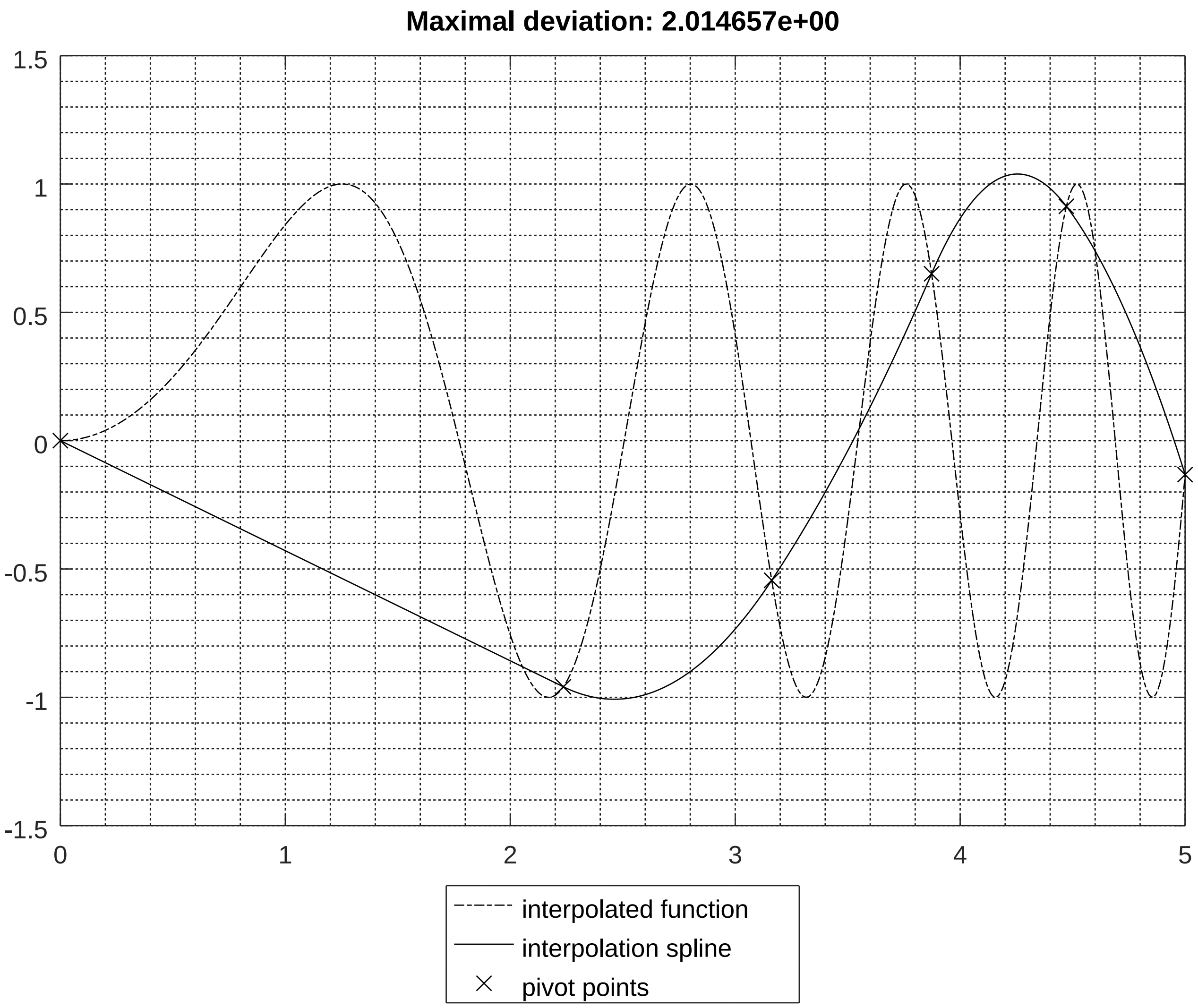
Формування коефіцієнтів сплайну.

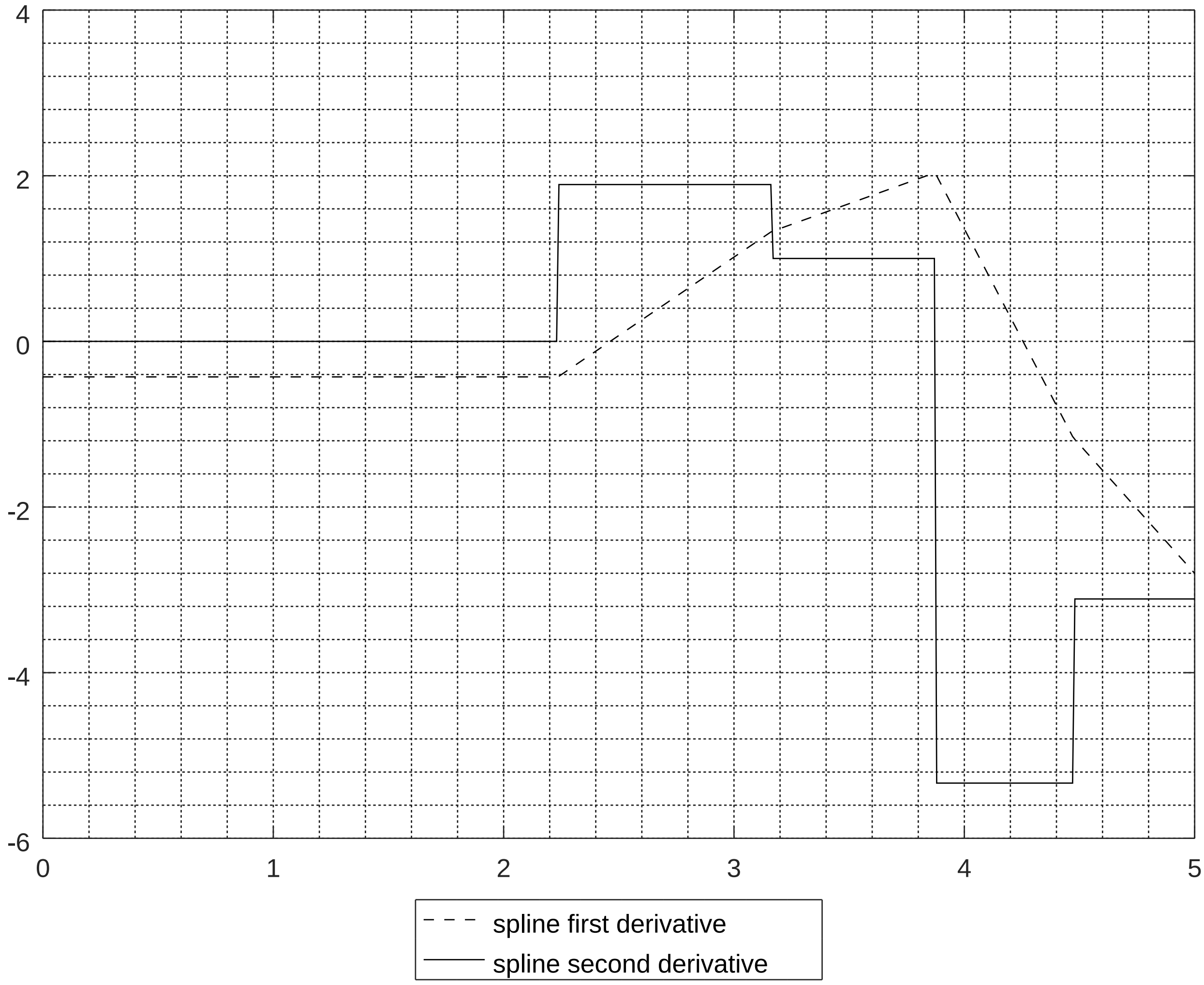
FormSpline.m

```
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, deltas - segments .* solution, values(1 : end - 1)];
end;
```

Нижче наведений результат роботи програми:


Maximal deviation: 2.014657e+00
1.5
1
0.5
0
-0.5
-1
-1.5
0
1
2
3
4
5
interpolated function
interpolation spline
pivot points



4
2
0
-2
-4
-6
0
1
2
3
4
5
spline first derivative
spline second derivative