REPORT: TASK 1

COMPOSITE 2-POINT GAUSS-LEGENDRE

Halil Suheyb Becerek
becerekh@student.mini.pw.edu.pl
295448

December, 2021

Attachments:

- importfile.m
- getData.m
- GaussLegendreTests.m
- verifyEqual.m
- mIntervalGaussLegendre.m
- GaussLegendre.m

1 Task Description

17. Numerical calculation of the integral $\int_a^b f(x)dx$ with given accuracy δ. Use the composite 2-point Gauss-Legendre rule. Start with calculating the approximation to the integral by the composite 2-point Gauss-Legendre rule where the interval of integration is partitioned into m subintervals. Then double the partition (take $2m$ subintervals), calculate the approximation to the integral with $2m$ subintervals, and calculate the absolute value of the difference between these 2 approximations. Then then double the partition again (take $4m$ subintervals), and again, and again, if necessary, until the absolute value of the difference between two consecutive approximations is smaller than δ.

2 Methodology

Gaussian quadrature rule applied individually to every sub-interval of the integration interval, It is called composite Gaussian quadrature.

Let

$$\int_a^b f(x)\,dx = \sum_{k=0}^{n} w_k f(x_k) \quad (1)$$

integral to be evaluated. Suppose this interval is $[a, b]$ split into m sub-intervals with equal width d then:

$$d = \frac{b-a}{m} \quad (2)$$

Gauss Legendre Rule requires that integration limits of any sub-interval needs to be transformed to interval [-1,1] then summation formula applied to each sub-interval with coefficients is:

$$\int_{x_{i-1}}^{x_i} f(x)\,dx = \frac{d}{2}\int_{-1}^{1} f(\frac{d}{2}z + x_{i-0.5})\,dx \approx \frac{d}{2}\sum_{k=0}^{n} w_k f(\frac{d}{2}z_k + x_{i-0.5}) \quad (3)$$

where $x_{i-0.5}$ - midpoint of the ith sub-interval; z_k - kth node; w_k - kth coefficient

In matlab a vector of midpoint values could be obtained by starting from smaller Interval range a adding half of width d, $a + \frac{d}{2}$ (this is midpoint of first sub-interval) iterating with step size d - width of each sub-interval - till we get to the midpoint of last sub-interval $b - \frac{d}{2}$.Values of w and z are read from the given 'GL.dat' file for $n = 2$.

Result of Integral over the whole range is equal to sum of individual results of sub-intervals

$$\int_a^b f(x)\,dx = \int_{x_0}^{x_1} f(x)\,dx + \int_{x_1}^{x_2} f(x)\,dx + \cdots + \int_{x_{m-1}}^{x_m} f(x)\,dx \quad (4)$$

Switching each Integral on RHS of (4) to form of (3) gives us

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{m}\frac{d}{2}\sum_{k=0}^{n} w_k f(\frac{d}{2}z_k + x_{i-0.5}) \quad (5)$$

3 Algorithm Description

Since the methodology was described in the section before, now top to bottom approach will be taken while describing the Algorithms in 'GaussLegendre.m', 'mIntervalGaussLegendre.m'

3.1 currentResult = GaussLegendre(f,m,d,a,b)

function takes 5 arguments

- f - Handle to function to integrate
- m - Initial number of sub-intervals
- d - Boundary on error
- a - lower bound on range
- b - upper bound on range

Import the file into single column table with values then parse the n,coefficients and nodes into n cell arrays

```
GL = importfile('GL.dat');
data = getData(GL);
```

Unpack the values of n, coefficients and nodes values from data (vector of cell arrays) for n = 2

```
        [n, coefficients, nodes] = deal(data{1,:,:});
```

Initialize consecutive results, reason of such initialization is to have minimum criteria to pass loop condition

```
        prevResult = 0;
        currentResult = d;
```

Iterate while error is smaller than the bound, make currentResult from previous iteration prevResult in this and calculate currentResult using mIntervalGaussLegendre(f,m,coefficients,nodes,a,b), double the m.

```
    while abs(currentResult - prevResult) >= d
        prevResult = currentResult;
        currentResult = mIntervalGaussLegendre(f,m,coefficients,nodes,a,b);
        m = 2*m;
    end
```

3.2 I = mIntervalGaussLegendre(f,m,coefficients,nodes,a,b)

function takes 6 arguments:

- f - Handle to function to integrate
- m - Number of sub-intervals
- coefficients - Coefficients/Weights of 2-point rule
- nodes - Nodes of 2-point rule
- a - lower bound on range
- b - upper bound on range

Initialize I which will be result of integral, Calculate width d of sub-intervals and midPoint for each sub-interval.

```
    I = 0;
    d = (b-a)/m;
    midPoint = [a + d/2:d:b-d/2];
```

Iterate from 1 to m:

1. Calculate $x = \frac{d}{2}z + x_{i-0.5}$ returning x will be 1x2 vector.
2. Evaluate $y = f(x)$ since x is a vector resulting y also will be a vector.
3. Calculate $I = I + \frac{d}{2}w_k y$ realize that both w_k and y are 1x2 vectors making use of the transposes allow us to utilize the dot product which calculates 2-point rule $w_1y_1 + w_2y_2$.Since Integration of the whole interval is sum of Integration of the sub-intervals we add to I in each step.

```
    for i=1:m
        x = midPoint(i) + nodes'*d/2;
        y = f(x);
        I = I + d/2*(coefficients'*y');
    end
```

4 Testing Framework

In order to allow easy testing, two functions were created 'GaussLegendreTests' and 'verifyEqual'.

GaussLegendreTests function takes no arguments and it is the place test cases reside and can be extended, to check the correction of results Matlab's own Integral function was used.

```
    function GaussLegendreTests()
        f = @(x)(x);
        actual = GaussLegendre(f,2,10.^-6,-1,1);
        expected = integral(f,-1,1);
        verifyEqual(f,actual,expected,-1,1,10.^-6);

        f = @(x) (x.^2);
        actual = GaussLegendre(f,2,10.^-6,-1,1);
        expected = integral(f,-1,1);
        verifyEqual(f,actual,expected,-1,1,10.^-6);

        f = @(x) (x.^2 + x);
        actual = GaussLegendre(f,2,10.^-6,-1,1);
        expected = integral(f,-1,1);
        verifyEqual(f,actual,expected,-1,1,10.^-6);

               .
               .
               .
    end
```

Not whole file, please check actual file to see all tests

verifyEqual(f,actual,expected,a,b,d) checks whether the expected result and actual result are accurate in the error range and outputs the results to the console

```
function verifyEqual(f,actual,expected,a,b,d)
    if(abs(actual - expected) < d)
        fprintf('[expected:] %f got [actual:] %f evaluating %s on [%d,%d]\n',
    expected,actual,functions(f).function,a,b);
    else
        fprintf('[!!!][expected:] %f got [actual:] %f evaluating %s on [%d,%d]\n',
    expected,actual,functions(f).function,a,b);
    end
end
```

5 Test Results

```
>> GaussLegendreTests()
[expected:] -0.000000 got [actual:] 0.000000 evaluating @(x)(x) on [-1,1]
[expected:] 0.666667 got [actual:] 0.666667 evaluating @(x)(x.^2) on [-1,1]
[expected:] 0.666667 got [actual:] 0.666667 evaluating @(x)(x.^2+x) on [-1,1]
[expected:] -0.000000 got [actual:] 0.000000 evaluating @(x)(sin(x)) on [-1,1]
[expected:] 1.682942 got [actual:] 1.682942 evaluating @(x)(cos(x)) on [-1,1]
[expected:] 1.809048 got [actual:] 1.809048 evaluating @(x)(cos(x.^2)) on [-1,1]
[expected:] 2.429585 got [actual:] 2.429585 evaluating @(x)(cos(x.^2)+sin(x.^2))
    on [-1,1]
[expected:] 2.925303 got [actual:] 2.925303 evaluating @(x)(exp(x.^2)) on [-1,1]
[expected:] 0.735759 got [actual:] 0.735759 evaluating @(x)(x.*exp(x)) on [-1,1]
[expected:] 0.602337 got [actual:] 0.602337 evaluating @(x)(x.*sin(x)) on [-1,1]
[expected:] 1.741591 got [actual:] 1.741591 evaluating @(x)(x.*sin(x)) on [0,2]
```