Technion – Israel Institute of Technology



HW7

Numerical Methods

019003

|  |  |  |
| --- | --- | --- |
| Alon Spinner | 305184335 | alonspinner@gmail.com |
| Oren Elmakis | 311265516 | orenelmakis@gmail.com |

January 24, 2022

# Question 1: Gauss-Quadrature with 3 sampled points

Given points to sample from, we will construct a polynomial of degree .

Given an integral in bracket , one can introduce a variable change such that after, values in it will range from

As such, we will only practice on the integral in bracket .

We assume the integral could be written as a sum of weights:

Integrating over the polynomial (left side):

Substituting the polynomial values in the sum (right side):

Comparing left side evaluation to the right-side evaluation:

Comparing Coefficients:

We have a system of equations – 6 equations and 6 variables .

Using MATLAB’s symbolic solver to write a general script for points.  
The results for are as follows:

Table GQ coefficients for M=3

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

# Question 2: Solving

In this question we will consider the bracket , computing the integral over with points.

## Analytical Solution:

We used MATLAB to solve the symbolic integral:

## Closed Newton-Cotes with 3 points

Closed Newton-Cotes is the Simpson method

## Open Newton-Cotes with 3 points

## Gauss-Quadrature with 3 points

From Question we know that:

Computing from :

To avoid losing accuracy to digit precession, we use the long format of double for presentation.

## Comparing the methods

Comparing the results, we produced the following table:

Table Method Comparison

Graphical user interface, application

Description automatically generated

The first thing that’s visible right off the bat, is that GQ has the smallest error by a large margin, following NC-open

Having NC-open provide better result than NC-closed is not assured, but also not surprising, as both methods have their errors proportional to (assuming 3 points), but the step size in NC-open is smaller.

Additionally, we have seen in the lectures that the GQ method can integrate a polynomial of degree exactly, while the methods can do the same for a polynomial of degree . As such, it is of no surprise, that it outperforms them.

We decided to plot the function and the integration points for further insight, presenting the figure below:

Chart, line chart

Description automatically generated

Figure Method Comparison on Axes

The integration bracket, the function’s derivative is monotonically increasing. As such, having well placed integration points near the higher end of the bracket will result in a better integration estimation.  
Unfortunately, we weren’t able to obtain any further intuition from the figure.

One final note:  
We decided against evaluating and comparing the computation time for each method, as they are all . In the lecture we were presented with the fact that the GQ method is more efficient.

# Question 3: Adaptive Gauss Quadrature

We were asked to implement the function *AdaptQuad(fun,a,b,n,epsilon)* which returns the integral approximation *I*, of function *fun* (sometimes denoted *‘f’*), over bracket *[a,b]* given *n*, number of sample points per section, and *epsilon*, the error tolerance for section subdivision.

Use *epsilon* =

In “Numerical Methods for Engineers” 6th addition, we are given the error term for the GQ method in equation . It shows as it appears below:

A picture containing diagram

Description automatically generated

Where

We could think of two methods to implementing an *adaptive gauss quadrature*.

1. Use the error term to develop relationship between the error of and , increasing until the division between them will be less than *epsilon.*In this method, one could solve for a sufficient without integrating once.

We have let MATLAB do the job and found the following relationship under the assumption that the derivative term in the error remains similar and can be cancelled

Chart, line chart

Description automatically generated

Figure GQ rational between consecutive error terms

Equating the rational to *epsilon* and solving numerically via *vpasolve* provides the answer of .

1. Use the same halving technique used in the common NC adaptive quadrature, which is developed using *Boole’s rule*, applying GQ integration instead of NC integration within it.  
   As we have no Boole’s rule for GQ, our best integral estimate will be

is the number of brackets after all subdivisions have been performed.

We stop subdividing in bracket when the relative error is smaller than *epsilon*

We decided to implement method 2, and provide the pseudo-code used as a reference from “Numerical Methods for Engineers” 6th addition figure 22.5 below.

We also changed the header of the function adding an optional argument which makes our recursive function more efficient when it must test whether to subdivide or not.

Table Adaptive Newton Cotes from 'Numerical Methods for Engineers'

A screenshot of a computer

Description automatically generated with medium confidence

## Solving

The figures below plot computation time and absolute error terms for different choices of . For each we computed the integral estimate with our adaptive GQ 10 times for a better estimate of computation time via averaging. The integral estimate itself is deterministic.

We would note that the MATLAB’s integral is not a function of , and was also computed 10 times for an average estimate, with the same relative error *epsilon*.

The absolute error from was computed using MATLAB’s symbolic solver.

We started from as using only one integration point is inadequate for any method.  
After-all, Integration solves for area given length.

Chart, line chart

Description automatically generated

Figure 3 Computation Time and Error analysis for f(x)= exp(3x)sin(2x)

First thing to notice is that the time computation decreases as we increase , which works well with our recursive method, as a lower initial will require our algorithm to create more subdivisions.

Second thing is, that after , the *GQ* evaluations keep at the value 3, which means that rational error term was small enough that we exited after the first function call.  
Additionally, we saw that for a reasonable number we reach the same computation error as MATLAB’s method, for a cheap computational cost.

It is worth mentioning that if we increase higher and higher, then our algorithm will produce the same integral approximation, but for a much higher computational price, stemming from the nonlinear optimization problem that is solved to find the *GQ* coefficients, as for there are always only 3 *GQ* evaluations.  
This can be seen in the figure below:

Graphical user interface, chart

Description automatically generated with medium confidence

Figure f(x)= exp(3x)sin(2x) analysis for high n numbers

## Solving

We used the exact same computation and averaging technique as in segment (a) of this question.Chart, line chart

Description automatically generated

Figure Computation Time and Error analysis for f = xsin(x^2)

Unlike the function in section (a), we have an increase in the number of *GQ* evaluations between and .  
This spike is caused by ‘luck’. As it happens, the *GQ* evaluation points land just right when to balance the negative and positive errors. Intuition for that can be provided by looking on the figure below.

Chart, line chart

Description automatically generated

Figure n=2 vs n=3 for f(x)=x\*sin(x^2)

Other than this fact, the results don’t add or deduct from our previous analysis in section (a), and for that reason we will refer you there.