Numerical Integration

Vidar Lunde Olaisen
https://github.com/VidarLuOl/Prosject3.git

October 22, 2019

Abstract

Through this assignment i showed that the brute force algorithms of Gauss Quadrature and Monty Carlo are more time efficient than the improved versions. The improved versions on the other hand requires less points to approximate the correct integral accurately.

1 Introduction

The meaning of all Integration methods is to manage to approximate the integral by the following formula

$$\mathbf{I} = \int_b^a f(x)dx \approx \sum_{i=1}^{N} w_i f(x_i)$$

This also includes the Gauss quadrature, especially Gauss Quadrature's special differential equations Gauss-Laguerre quadrature and Gauss-Legendre quadrature. We will try these and also Monte-Carlo integration to integrate a 6-dimensional integral that tells us the energy between two electrons in a helium atom.

The integral for the two electrons that repels each other is the classic Coulomb interaction which is as follow

$$\langle \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|} \rangle = \int_{-\infty}^{\infty} d\mathbf{r}_1 d\mathbf{r}_2 e^{-2\alpha(r_1+r_2)} \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|}$$

These are however not the only ways of finding the integral but they are the one we will focus on in this assignment.

2 Theory

2.1 Gauss quadrature

Normally when we try to approximate an polynomial with N points we use an integration function that uses the Taylor series and will generate a polynomial of exactly N-1 degree. They will then choose the weight to satisfy the linear equations best. This is where Gauss Quadrature differs, as it finds the weights through the usage of orthogonality where the points are selected in the interval made. We can then use the 2N parameters to more efficiently approximate the integration.

$$\int f(x)dx \approx \int P_{2N-1}(x)dx = \sum_{i=0}^{N-1} P_{2N-1}(x_i)w_i$$

While the given integrand might not be the smoothest function we can rewrite the equation by removing the weight function from the orthogonal polynomial and instead work with a given weight.

$$\mathbf{I} = \int_b^a f(x)dx = \int_b^a W(x)g(x)dx \approx \sum_{i=1}^{N} w_i f(x_i)$$

The Weight in the function works in a way that certain parts of the function is more worth and others, and will therefore be given bigger weight value to empathize this.

2.1.1 Gauss-Legendre

The Gauss Legendre Quadrature is a numerical integration can be called the standard Gauss Quadrature. What identifies it is that it spans across an intervall from -1 to 1 and has a weighted function W(x) = 1.

$$\mathbf{I} = \int_{-1}^{1} f(x)dx = \int_{-1}^{1} W(x)g(x)dx \approx \sum_{i=1}^{N} w_i f(x_i)$$

$$W(x) = 1$$

$$x = [-1, 1]$$

2.1.2 Gauss-Laguerre

Gauss-Laguerre Quadrature is also a numerical integration that can be said to be an extension of the Gauss Quadrature. It's used to gain an efficient and accurate numerical valuation of the values given. It's known features is that its limits goes from zero to ∞ and that its weight is e^{-x^2} fs

$$W(x) = e^{-x^2}$$

$$x = [0, \infty]$$

2.2 Monte-Carlo integration

Compared to Gauss Quadrature the Monte-Carlo integration uses non-determined values on it's integration to find the right value and to model the uncertainty of a model. The reason of it's popularity is because its so easy to compute and read. It takes an span between 0 and 1, and put's all the random values into it and according to statistics it should spread out evenly

$$\int f(x)p(x)dx \approx 1/N \sum_{i=1}^{N} f(x)$$

$$\int_0^\infty x^\alpha e^{-x} dx$$

The variance of the MonteCarlo Integration can be explaned as how large one might expect the fault to be if we do the experiment several times. This is where the randomness factor of Monte Carlo comes inn, becuase it's random one is almost never going to get the same answer twice and we therefore have to take into conclusion that the values will move within a space that's size is defined by the variance.

$$\sigma^2 = \int p(x)f^2(x)dx - \big[\int f(x)p(x)dx\big]^2$$

$$\approx 1/N \sum f^2(x_i) - \big[1/N \sum f(x_i)\big]^2$$

3 Method

3.1 Algorithm and explanation

3.1.1 Gauss-Legendre Quadrature

Algorithm 1 Gauss-Legendre Algorithm

```
Roots = (n + 1)/2

eps = 10^-4
for i <= m do
    We start with an approximation to the ith root

    for j <= n do
        We get the Legendre polynomial evaluated at x
    end for

    We now got the desired Legendre Polynomial
end for

while fabs(z - z1) > ZERO do
    Scales the root to the desired interval
    Computes weight and it's symmetric counterpart
```

This is the brute force way of integrating the integral with Gauss-Legendre that works but isn't very efficient with larger data.

3.1.2 Improved Gauss-Quadrature

Algorithm 2 Gauss-Legendre Quadrature Algorithm

```
for i1 < N do (
    for i2 < N do
        for i3 < N do
            for i4 < N do
                for i5 < N do
                    for i6 < N do
                        Calculates Gauss by checking the weight and value of each variable.
                        Uses Gauss Legendre Algorithm.
                    end for
                end for
            end for
        end for
    end for
end for=0
```

To improve the Gauss-Quadrature we try to change the coordinate frame to polar-coordinates, we therefore have to rewrite the Coulomb interaction from

$$\langle \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|} \rangle = \int_{-\infty}^{\infty} d\mathbf{r}_1 d\mathbf{r}_2 e^{-2\alpha(r_1+r_2)} \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|}$$

to

$$\langle \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|} \rangle = \int_{-\infty}^{\infty} r_1^2 dr_1 r_2^2 dr_2 dcos(\theta_1) dcos(\theta_2) d\phi_1 d\phi_2 e^{-2\alpha(r_1+r_2)} \frac{1}{\sqrt{r_1^2 + r_2^2 - 2r_1 r_2 cos(\beta)}}$$

3.1.3 Monte Carlo

Algorithm 3 Monte Carlo Algorithm

```
Sum = 0
Variance = 0

for i < N do
    gives all the 6 variables a random value
    Integrates them all with Legendre
    Adds the value from integration into sum and value^2 into variance
end for

Divides the total sum with total points, same with variance
```

3.1.4 Improved Monte Carlo

Algorithm 4 Improved Monte Carlo Algorithm

```
Sum = 0
Variance = 0

for i < N do
    gives all the 6 variables a random value
    Integrates them all with Legendre
    Adds the value from integration into sum and value^2 into variance
end for

Divides the total sum with total points, same with variance
return sum*jacobi
```

The code that i tried didn't work and i did not manage to find out why. But to improve the Monte Carlo algorithm one must do the same as in the improvement of the Gauss Quadrature and convert it from cartesian coordinates to polar coordinates. This changes thing like the jacobi variable from

$$Jacobi = (lambda * 2)^6$$

to

$$Jacobi = (pi * pi * 2pi * 2pi)^6$$

3.2 Testing

To verify that the implemented functions work as expected and does not change when we change the programs, we implemented unit test. Unit tests are most useful when implemented early in the process of creating a new algorithms, to help finding the source of errors.

3.2.1 Testing the Gauss-Legendre Quadrature

The test for finding out if the Gauss-Legendre Quadrature works takes its conclusion from that the provided answer $5\pi^2/16^2$ is the correct approximation. I therefore try to compare the two values, the first is the one I've calculated thanks to the Gauss-Legendre integration and the second is the answer given. The Approx function will then see if the results are equal by an variance of 0.01 as it's the requested scale that the task asks you to look for.

3.2.2 Testing the improved Gauss-Quadrature

This test has the same premise as the first test but in this case its in polar coordinates instead of cartesian coordinates. We therefore use a combination of the Gauss-Legendre and Gauss-Laguerre to get the wanted result. When we compare the calculated result we can see that we get wrong answer, but that's because the formula in theory should work, but in the computing iterates towards a value a bit above the correct value. We therefore get an error in the test.

3.2.3 Testing the Monte Carlo Integration

The testing of the Monte Carlo integration was just to compare the result from the Montecarlo function and the exact value. We're using 2.3 as lambda.

4 Results

4.1 Finding the integral by Gauss-Legendre quadrature

The plot that we made out from the e^x where we found out where the optimal value of λ

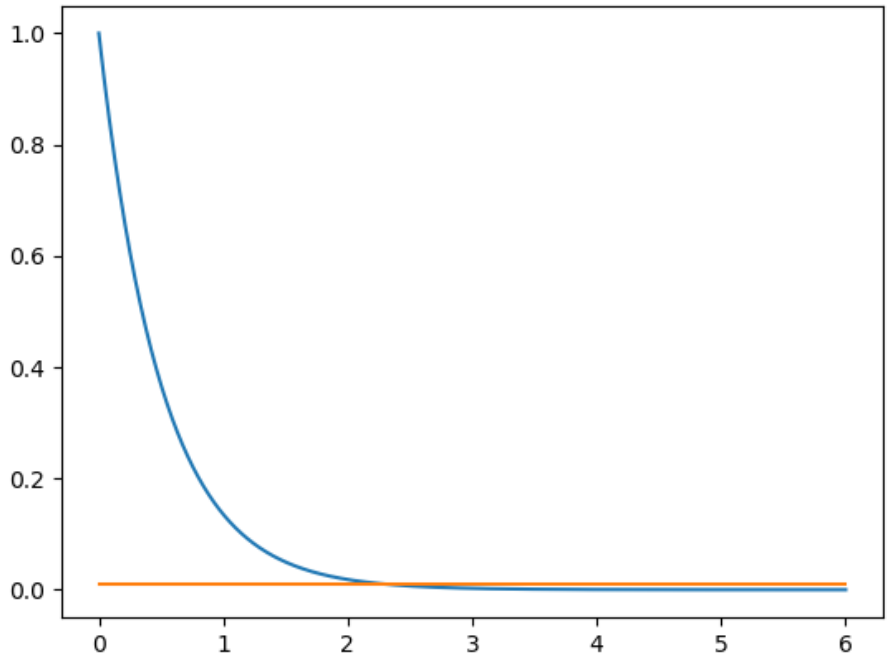

Figure 1: The two lines cross eachother at around 2.3, where the blue line(e^x) goes below the selected lowest acceptable value.

When we've found out the value of lambda we can start cheking for what values of mesh points(N) we need for the result to start converging at a level 3 point digit. In this case it's 17.

Each of the following values are given after one try, so if any one of them can be an outliner but it's expected that the abs(dif) will decrease for each time we raise N, which means the computation code works.

N	Time	Approximation	Difference
15	0.883914	0.231789	0.0390229
19	3.76944	0.207583	0.014817
23	13.0111	0.196947	0.00418153
27	29.7738	0.192347	-0.000418532

4.2 Improved Gauss-Quadrature

Each of the following values are given after one try, so if any one of them can be an outliner but it's expected that the abs(dif) will decrease for each time we raise N, which means the computation code works.

N	Time	Approximation	Difference
15	2.98648	0.195327	0.00256176
19	11.102	0.201607	0.00884101
23	32.9138	0.205023	0.0122572
27	86.4491	0.207072	0.0143063

As you can see the Difference is increasing even though it should decrease, this is not because of the theory behind the code, but rather an computational error that makes the Quadrature integrate towards a point that has higher value than it really does. Therefore as you can see it integrates towards an value around 0.207072.

4.3 Monte-Carlo integration

Each of the following values are given after one try, so if any one of them can be an outliner but it's expected that the abs(dif) will decrease for each time we raise N, which means the computation code works.

N	Time	Approx	Variance
10	0.000095895	0.00129407	9.72882e-10
100	0.000200984	4.33665	0.00171311
1000	0.000660922	0.207965	0.00149675
10000	0.00802139	0.154049	0.00205346
100000	0.0616715	0.156046	0.00415939
1000000	0.673902	0.177935	0.00731558
10000000	6.07566	0.196448	0.0296956

4.4 MPI of Monte-Carlo

Each of the following values are given after one try, so if any one of them can be an outliner but it's expected that the abs(dif) will decrease for each time we raise N, which means the computation code works.

2 processors

N	$time_n2$	$Approx_n2$	$diff_n2$
10	0.00126042	0.000463401	-0.192302
100	0.00765424	0.0540132	-0.138753
1000	0.0116064	0.115028	-0.0777379
10000	0.115264	0.176259	-0.0165065
100000	0.490108	0.200418	0.00765218
1000000	0.315325	0.192225	-0.000540768
10000000	2.90898	0.96696	0.00393051

3 processors

N	$time_n3$	$Approx_n3$	$diff_n3$
10	0.00825772	0.000943753	-0.191822
100	0.0106498	0.113159	-0.079607
1000	0.00891137	0.103609	-0.0891571
10000	0.0116357	0.164151	-0.0286147
100000	0.0404794	0.158492	-0.0342733
1000000	0.23313	0.191914	-0.000851264
10000000	2.15295	0.193105	0.00033889

4 processors

N	time$_n$4	Approx$_n$4	diff$_n$4
10	0.0122351	0.43701	-0.14906
100	0.0274982	0.0298844	-0.162881
1000	0.0132979	0.0354807	-0.157285
10000	0.0229455	0.307732	0.114967
100000	0.0407729	0.168853	-0.0239124
1000000	0.23608	0.17938	-0.013386
10000000	1.92223	0.189959	-0.00280689

As one can see the accuracy of the algorithm does not suffer at the expense of more processors, but it will go faster.

4.5 Improved Monte-Carlo integration

The Improved Monte-Carlo integration code that i made didn't work as expected and i could not figure out why, neither could the persons I've asked.

5 Discussion

The different quadratures have a way of finding the approximate integer but they work at a different pace. The brute force of the Gauss-Legendre Quadration will always be faster than the more delicate improved version that contains Gauss-Laguerre. Simply because it requires much less processing power from just having to involve one function compared to the three from the improved version. It does however need more points to get an accurate approximation to the correct integration value. As told the improved Gauss Quadrature does however contain an error that affects the wished result and therefore makes the quadrature integrate to a wrong value.

Monte Carlo works best the more points we have, this is obvious as the more samples we have the more certain it is that the points generated the less risk it has for errors. Because of the way it works it has a high chance of missing totally at low level of points as the numbers are randomly generated. So for it at all to be viable we need huge numbers of points.

The parallelization of the code will also work perfectly in these types of assignments as the computation isn't dependent on other variables being finished. We can therefore paralyze it without worrying about the nodes using variables that haven't been calculated yet.

6 Conclusion

From the results that I've achieved i can see a pattern that tells me that the brute force algorithm works faster when it comes to the calculation with given points in Monte Carlo or loops in Gauss Quadrature. This is because of the lower usage of math functions and generally less calls to functions. This does however go on the cost of accuracy of the approximation and it needs more points to get a good answer, so it would work good at big data but rather ineffective at small data. This is where the improved versions shine as it doesn't need as many point because of it's conversion to polar coordinates.

References

[1] Hjorth-Jensen, M. (2015). *Computational Physics - Lecture Notes 2015* . University of Oslo Albert Einstein.