UNIVERSITY OF OSLO

COMPUTATIONAL PHYSICS

# Project 3

UiO : University of Oslo


Authors:

Birgitte Madsen

Magnus Isaksen

Soumya Chalakkal


Autumn 2015

UiO : University of Oslo

**Department of Physics**
**University of Oslo**
Sem Sælands vei 24
0371 Oslo, Norway
+47 22 85 64 28
http://www.mn.uio.no/fysikk/english/

**Course:**

Computational Physics

**Project number:**

3

**Link to GitHub folder:**

`https://??`

**Hand-in deadline:**

Friday, October 23, 2015

**Project Members:**

Birgitte Madsen
Magnus Isaksen
Soumya Chalakkal

**Copies:** 1
**Page count: ??**
**Appendices:** 0
**Completed:** ??

# TABLE OF CONTENTS

CHAPTER

# 1 INTRODUCTION

The project mainly discusses about various integration methods namely Gaussian quadrature methods and Monte carlo methods that can be employed to solve an integral function. The integral we are interested in is the quantum mechanical expectation value of the correlation energy between two electrons which repel each other via the classical Coulomb interaction. In the Gaussian quadrature method two important orthogonal polynomials Legendre and Laguerre are used to evaluate the integral. Since Legendre polynomial is defined in the interval [-1,1] solving the integral gave unsatisfactory results. To improve the results the integral which was in cartesian coordinate is converted to spherical coordinates with different integration limits and Laguerre polynomial is introduced which is defined in the interval [0,infinity]. A comparison of the results using the two different polynomial is made. In order to employ Monte carlo methods we assumed the system can be described as a probability density function and random numbers were generated to cover uniformly in the interval [0,1]. First a brute force approach is used to find the solution then the method is improved with importance sampling. An error estimation is made in both cases. A comparison of all the method that we used to solve the integral is done in the end.

This report mainly consists of .. ?????

CHAPTER

# 2 METHOD

In this chapter a short introduction of the nature of problem is given together with a brief analysis of the function whose integral is to be calculated is made. Furthermore, the Legendre ans Laguerre method for computing the polynomial Laguerree method for computing the olynomial ????

## 2.1 Nature of the problem

Even though the Schrö dinger equation cannot be solved exactly for the helium atom or more complicated atomic or ionic species due to electron-electron interaction, the ground state energy of the helium atom can be calculated using approximate methods. One method is to assume that the electrons in helium atom occupies scaled hydrogen 1s orbital so that the product of the wave function of the two electrons can be given as

$$\Phi(r_1, r_2) = exp[-\alpha(r1 + r2)] \tag{2.1}$$

in which $\Phi_{1s}(r_i) = exp(-\alpha(r_i))$ is the single particle wave function for a particle at position $r_i$. $\alpha$ is a parameter that corresponds to the charge of helium atom and $r_i = \sqrt{x_i^2 + y_i^2 + z_i^2}$ is the cartesian coordinate of particle The ground state correlation energy between these two electrons in the helium atom can be calculated by solving the integral

$$\left\langle \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|} \right\rangle = \int_{-\infty}^{\infty} \frac{e^{-2\alpha(r_1+r_2)}}{|\mathbf{r}_1 - \mathbf{r}_2|} d\mathbf{r}_1 d\mathbf{r}_2 \tag{2.2}$$

which is the quantum mechanical expectation value of the correlation energy between two electrons which repel each other via the classical coulomb interaction. The closed-form solution of Eq. (2.2) is $5\pi^2/16^2$. ??????? Closed form solution is there in the below link but I didnt understand it properly. `https://www.physics.ohio-state.edu/~ntg/6810/readings/hjorth-jensen_notes2013_14.pdf` [1]

### 2.1.1 Integral Written in Cartesian and Spherical Coordinates

Writing out Eq. (2.2) in cartesian coordinates in which

$$\mathbf{r}_i = x_i\hat{\mathbf{i}} + y_i\hat{\mathbf{j}} + z_i\hat{\mathbf{k}} \tag{2.3}$$

[1] FiXme Note: write closed form solution ??

with $\hat{\mathbf{i}}$, $\hat{\mathbf{j}}$ and $\hat{\mathbf{k}}$ being unit vectors in the x-, y- and z-direction, respectively and $x_i, y_i, z_i, \in (-\infty;\infty)$, yields

$$\left\langle \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|} \right\rangle = \int_{-\infty}^{\infty} \frac{e^{-2\alpha\left(\sqrt{x_1^2+y_1^2+z_1^2}+\sqrt{x_2^2+y_2^2+z_2^2}\right)}}{\sqrt{(x_1-x_2)^2+(y_1-y_2)^2+(z_1-z_2)^2}} dx_1 dy_1 dz_1 dx_2 dy_2 dz_2 \tag{2.4}$$

with the integral being from $-\infty$ to $\infty$ over all six variables. This representation of the integral will be used in the Gauss-Legendre method for computing the integral. The following lines of code shows how to compute the function to be integrated in c++.

```
double int_function(double x1, double x2, double x3, double y1, double y2, double y3)
{
  int alpha = 2.0;
  double denominator = pow((x1-y1),2)+pow((x2-y2),2)+pow((x3-y3),2);
  double exponent = exp(-2.0*alpha*(sqrt(x1*x1+x2*x2+x3*x3)+sqrt(y1*y1+y2*y2+y3*y3)));
  if (denominator < pow(10,-6)){return 0;}
  else return exponent/sqrt(denominator);
}
```

A problem arises if the denominator becomes very small, when computing the fraction, since this would cause the function value to become very large. To avoid this problem, the function value is set equal to zero if this is the case. For the Laguerre method, spherical coordinates with $r_i \in [0;\infty)$, $\theta_i \in [0;\pi]$, and $\phi_i \in [0;2\pi]$ are used. When writing the considered integral in spherical coordinates, the problem to solve becomes

$$\left\langle \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|} \right\rangle = \int \frac{e^{-2\alpha(r_1+r_2)}}{\sqrt{r_1^2+r_2^2-2r_1r_2cos(\beta)}} r_1^2 r_2^2 sin(\theta_1) sin(\theta_2) dr_1 dr_2 d\theta_1 d\theta_2 d\phi_1 d\phi_2 \tag{2.5}$$

with the integral having the limits as described above. $cos(\beta)$ is then given by

$$cos(\beta) = cos(\theta_1)cos(\theta_2) + sin(\theta_1)sin(\theta_2)cos(\phi_1 - \phi_2) \tag{2.6}$$

## 2.2 Gauss-Legendre Method for Computing the Integral

To be able to use the Gauss-Legendre method to compute the integral in Eq. (2.2), the limits of the integral must be made finite. Since the wave function

$$e^{-2\alpha x} \tag{2.7}$$

rapidly goes toward zero as $x$ is increased (see Fig. 2.1), the integral can be approximated by the same integral with finite limits.

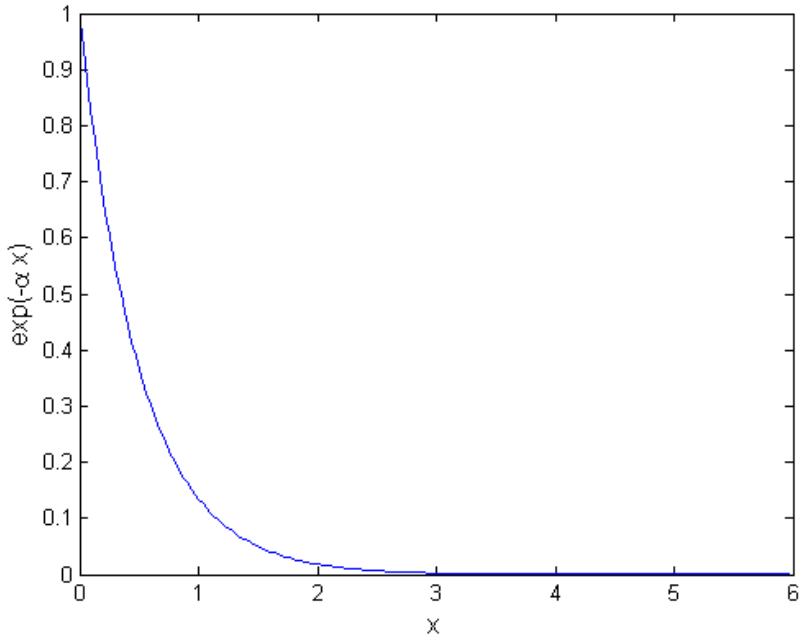


***Figure 2.1.*** Plot of $e^{-2\alpha x}$ with $\alpha = 2$.

[2]

In this project, if the value is equal or smaller than $10^{-6}$ it is close enough to zero to neglect contributions from the part of the wave function when the wave function gives a value of this order. For $x = 4$ the value of the wave function is $e^{-2\cdot\alpha\cdot 4} \approx 1.1\cdot 10^{-6}$, when $\alpha = 2$. Hence, the considered integral that is to be solved by the Gauss-Legendre method is given by

$$\left\langle \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|} \right\rangle = \int_{-4}^{4} \frac{e^{-2\alpha(r_1+r_2)}}{|\mathbf{r}_1 - \mathbf{r}_2|} d\mathbf{r}_1 d\mathbf{r}_2 \tag{2.8}$$

To be able to use the Gauss-Legendre method, the limits have to be $-1$ and 1. This is, however, easily obtained by a change in variables using the following quantity

$$\int_a^b f(t)dt = \int_{-1}^{1} f\left(\frac{b-a}{2}x + \frac{b+a}{2}\right) dx \tag{2.9}$$

The first step of the Gauss-Legendre method is then to compute the roots of the $n$'th Legendre polynomial. The $i$'th root $z_i$ is approximated by

$$z_i = \cos\left(\frac{\pi(4\cdot i - 1)}{4n+2}\right) \tag{2.10}$$

for large $n$. The $n$'th Legendre polynomial $L_n(x)$ can then be computed using the recursive relation

$$(j+1)L_{j+1}(z) + jL_{j-1}(x) - (2j+1)zL_j(z) = 0 \tag{2.11}$$

with $L_{-1}(x) = 0$ and $L_0(x) = 1$. The code for generation the Legendre polynomial can be found in sec ?? [3] together with a test of the Legendre polynomial generating algorithm.

[2] FiXme Note: fix figur
[3] FiXme Note: ref to sec

The derivative of the Legendre polynomial can then be computed as

$$L_n'(z) = \frac{-n \cdot z \cdot L_n(z) + n \cdot L_{n-1}(z)}{1 - z^2} \tag{2.12}$$

Newton's method is then applied to find the best estimation of the roots by considering the expression

$$z_1 = z - \frac{L_n(z)}{L_n'(z)} \tag{2.13}$$

in which $z$ is the first approximation of the root given by Eq. (2.10), and $z_1$ is a better approximation for the root. The algorithm for generation of the roots and the Legendre polynomials is run until the difference between $z_1$ and the root approximated by Eq. (2.10) is ultimately zero, which in this project is set to $10^{-6}$.

The weight $w_i$ of the $i$'th root $z_i$ is then obtained by

$$w_i = \frac{2}{(1 - z_i^2)(L_n'(z_i))^2} \tag{2.14}$$

## 2.3 Generation of Legendre Polynomials

The following lines of code generates the $n$'th order Legendre polynomial from the recursive relation given in Eq. (2.11) and its roots. Since the roots are symmetric about 0, the for-loop is run from $i = 1$ to $i < m$ for $m = n + 1$ with $n$ being the order of the computed polynomial and hence the length of the array containing the roots and the array containing the respective weights of the roots.

```
//Code for computing the Legendre polynomials to determine the roots of the n'th
    Legendre polynomial
    for (int i = 1; i<=m;i++){
        root = cos(pi * (4*i-1) / (4*n + 2 ));
        //approximation of the root of the n'th polynomial
    do{
        // This uses the recursive relation to compute the Legendre polynomials
        double L_plus = 1.0 , L = 0.0, L_minus;
        for (int j = 0; j < n; j++)
        {
        L_minus = L;
        L = L_plus;
        L_plus = (2.0*j +1)*root*L - j*L_minus ;
        L_plus /= j+1;
        }
      //derivative of the Legendre polynomial (L_plus)
        dev_L = (-n*root*L_plus + n*L)/(1-root*root);
        // Newton's method
        z = root;
        root  = z - L_plus/dev_L;
       } while(fabs(root - z) > pow(10,-6));
       //compute values of x (array containing the roots) and w (array containing the
           roots)
        * w_temp1 = 2/((1-root*root)*(dev_L*dev_L));
        *(x_temp1++) = root;
```

```
        *(x_temp2--) = -root;
        *(w_temp2--) = *(w_temp1++);
    }
```

The algorithm starts with an approximation of the $i$'th root of the $n$'th order Legendre polynomial as described in Eq. (2.10). After computation of the Legendre polynomial, the approximation is improved by Newton's method. If the difference between the improved approximation and the initial approximation is greater than zero (that is $10^{-6}$), the $n$'th order Legendre polynomial is computed using this new approximation, and with this new Legendre polynomial the previously improved approximation of the root is again improved, and once again, if the difference between the approximated roots is greater than zero, new Legendre polynomials are computed. This procedure is run until the difference between the approximation of the $i$'th root and the improved approximation of the $i$'th root is zero. When this is achieved, the root $z_i$ is put into the $i$'th and $(n-i)$'th entry of the array containing the roots, and the corresponding weights are computed by Eq. (2.14). This is done $m$, in which $m$ is the lowest integer value of $(n+1)/2$ times, and hence all the entrances of the arrays $x[n]$ and $w[n]$ will be filled with the roots and corresponding weights

### 2.3.1 Test of the Legendre Polynomial Generation

Tab. 2.1 provides the roots and corresponding weights of the $n$'th Legendre polynomial. [4] For $n = 5$, these are

***Table 2.1.*** Given roots and weights for the Legendre polynomial

| Roots | Weights |
|---|---|
| 0 | 0.568889 |
| $\pm$ 0.538469 | 0.478629 |
| $\pm$ 0.90618 | 0.236927 |

which is exactly what is obtained up to the same accuracy as in Fig. 2.1 by running the above lines of code for $n = 5$. (see [5])

## 2.4 Laguerre Method for Computing the Integral

In the Gauss-Laguerre method integrals of the form

$$I = \int_0^\infty x^\alpha e^{-x} g(x) dx \tag{2.15}$$

can be solved using Laguerre polynomials and a weight function given by

$$W(x) = x^\alpha e^{-x} \tag{2.16}$$

[4] FiXme Note: source
[5] FiXme Note: ref. to GitHub

Hence, if the variables of the integral in Eq. (2.2) are changed from cartesian coordinates to spherical coordinates with $r_i \in [0;\infty)$, $\theta_i \in [0;\pi]$, and $\phi_i \in [0;2\pi]$, the integral can be solved by using the roots and corresponding weights of Legendre polynomials for $\theta_i$ and $\phi_i$ and use roots and corresponding weights of Laguerre polynomials for $r_i$. The algorithm for finding the roots and their corresponding weights of the Laguerre polynomials will, however, not be discussed here. An important note is, though, that since the weight function includes $x^{\alpha}$ and $e^{-x}$, the function that is to be evaluated in the roots of the $n$'th Laguerre polynomial is

$$g(r_1,r_2,\theta_1,\theta_2,\phi_1,\phi_2) = \frac{e^{-2\alpha(r_1+r_2)+r_1+r_2}}{\sqrt{r_1^2+r_2^2-2r_1r_2cos(\beta)}} sin(\theta_1)sin(\theta_2) \tag{2.17}$$

with $cos(\beta)$ described by Eq. (2.6).

## 2.5 Monte Carlo Method

Monte Carlo methods are used for problems that can be described using probability distribution functions. This part will look at a brute force approach using two different probability distribution functions (PDF). With the probability distribution function the approach is to characterizing a problem using random numbers. Then getting a sum out of the function evaluated and considering the error made by finding the variance using this method.

### 2.5.1 Brute force method

The brute force method uses the uniform probability distribution function. Where the stochastic variables are approximated using random numbers. Where the probability distribution function gives a probability or the relative frequency for the stochastic variables. The probability function $Prob(X)$ is found by Eq. (2.18), where $p(x)$ is the probability for one of the variables.

$$Prob(a < X < b) = \int_a^b p(x)dx \tag{2.18}$$

The method consist in choosing a random number generator to make the stochastic values that goes in to the probability distribution function. This gives out the function values that make the integral sum. This also gives the possibility to generate the variance for estimating the error.

The integral sum is made using Eq. (2.19).

$$I = \frac{1}{N}\sum_{i=1}^{N} f(x) \approx < f(x) > \tag{2.19}$$

The variance using Eq. (2.20)

$$\sigma^2 = < x^2 > - < x >^2 \tag{2.20}$$

where $< x^2 >$ is the expectation value that is found from the given PDF using Eq. (2.21).

$$< x^k >= \frac{1}{N}\sum_{i=1}^{N} x_i^k p(x) \tag{2.21}$$

And for $< f(x) >$ it is found using Eq. (2.22)

$$< f(x) >= \frac{1}{N}\sum_{i=1}^{N} f(x)p(x) \tag{2.22}$$

Part of the code is included below, and it shows the for loop that generates the integral. It does this by generating a random number for all of the 6-dimensions that exists. These variables it sends to the function that is integrated, returning a function value that goes in to the sum generating the end integral. This is done N times, preferably with N as high as possible. After the for loop the sum is multiplied with the Jacobi determinant giving the end integral. Also the variance is generated by the function value and used to produce the standard deviation sigma.

```
for (int i=0;i<N;i++){
           for (int j=0;j<6;j++){
               x[j] = - length +2*length*((double) rand() / (RAND_MAX)); //random
                   numbers generated in the interval(-length,length)

               //cout<<x[j]<<endl;

           }

           fx = func(x[0],x[1],x[2],x[3],x[4],x[5]);
           Montec += fx;
           Montesqr += fx*fx;

       }

       double MC = Montec/((double) N );
       double Montesqr1= Montesqr/((double) N);
       double variance= Montesqr1-MC*MC;
       double volume = pow(length,6);
       cout << " Integral = " << jacobidet*MC << endl;
       cout << "sigma = " << jacobidet*sqrt(variance/ ((double) N)) << endl;
```

.

### 2.5.2 Improved Monto Carlo Method

In the improved brute force Monto Carlo method the coordinates of the function to be integrated are changed from cartesian coordinates to spherical coordinates with $r_i$ being exponentially distributed on the

interval $[0;\infty)$, $\theta_i$ being uniformly distributed on the interval $[0;\pi]$, and $\phi_i$ being uniformly distributed on the interval $[0,2\pi]$. That means that the PDF's of $\theta_i$ and $\phi_i$ are given by

$$p(\theta_i) = \frac{1}{\pi} \qquad \text{and} \qquad p(\phi_i) = \frac{1}{2\pi} \tag{2.23}$$

whilst the PDF for the radial component is given by

$$p(r_i) = exp(-r_i) \tag{2.24}$$

Since the considered function in spherical coordinates is given by Eq. (2.5), it is an advantage to do a change of the radial variables into

$$u_i = 2\alpha r_i \tag{2.25}$$

which is also exponentially distributed in the interval $[0;\infty)$ with the PDF fiven by Eq. (2.24). This change in variables could of cause also have been applied in the Laguerre method described in Sec. 2.4. This gives

$$du_i = 2\alpha dr_i \tag{2.26}$$

and hence the function that is to be evaluated becomes

$$f(u_1,u_2,\theta_1,\theta_2,\phi_1,\phi_2) = \frac{1}{(2\alpha)^5}\frac{e^{-u_1-u_2}}{\sqrt{u_1^2+u_2^2-2u_1u_2cos(\beta)}}u_1^2u_2^2sin(\theta_1)sin(\theta_2) \tag{2.27}$$

When dividing this function by the PDF's for the radial components $u_1$ ans $u_2$, it is evident that the exponential $e^{-u_1-u_2}$ will disappear. That means that when evaluating the integral with the Monte Carlo method, it becomes

$$I = \frac{1}{N}\sum_{i=1}^{N}\frac{\tilde{f}(u_i,\tilde{u}_i,\theta_i,\tilde{\theta}_i,\phi_i,\tilde{\phi}_i)}{p(\theta_i)p(\tilde{\theta}_i)p(\phi_i)p(\tilde{\phi}_i)} = \frac{1}{N}\sum_{i=1}^{N}\frac{\tilde{f}(u_i,\tilde{u}_i,\theta_i,\tilde{\theta}_i,\phi_i,\tilde{\phi}_i)}{4\pi^4} \tag{2.28}$$

in which $\tilde{f} = f/e^{-u_1-u_2}$.

The source code below shows the for loop for the improved Monte Carlo method can be seen below. The difference between this and the for loop for the brute force Monte Carlo method given in [6], is the definition of the random variables and explicit statement of the total PDF *px*.

```
for (int i=0;i<N;i++){
   for (int j=0;j<6;j++){
      x[j] = ((double) rand() / (RAND_MAX));
      //random numbers generated in the interval(0,1)
    }
      //making the random numbers for u_i, theta_i and phi_i on respective interval.
            double u1 = -log(1-x[0]);
            double u2 = -log(1-x[1]);
            double theta1 = x[2]*pi;
            double theta2 = x[3]*pi;
            double phi1 = x[4]*2*pi;
```

[6] FiXme Note: ref to sec

```
            double phi2 = x[5]*2*pi;
        //evaluating the function in the random u_i, theta_i and phi_i
            fx = func(u1,u2,theta1,theta2,phi1,phi2);
        //computing the product of the PDF's of theta_i and phi_i
            px = 1/(4*pi*pi*pi*pi);
        //computing the parts to be summed up
            Montec += fx/px;
            Montesqr += (fx/px)*(fx/px);
}
        //computing integral value and variance from which the standard deviation is
            calculated
        double integral_value = Montec/((double) N );
        double Montesqr1 = Montesqr/((double) N);
        double variance = Montesqr1-integral_value*integral_value;
        double standard_deviation = sqrt(variance/ ((double) N));
```

The function *func* that is called in the for loop is in spherical coordinates and with the change of variable $u_i = 2\alpha r_i$ as described in this section, unlike the function called in the for loop for the brute force Monte Carlo method, which is in cartesian coordinates. The random radial number $u_i$ is computed from the uniformly distributed random number $x_j$ in $[0;1]$ by the formula

$$u_i = -ln(1-x_j) \tag{2.29}$$

since the probability has to be conserved, yielding

$$p(u)du = exp(-u)du = dx = p(x)dx \tag{2.30}$$

in which $p(x) = 1$ since $x$ is uniformly distributed in $[0;1]$. Integration of Eq. (2.30) from 0 to $u$ yields

$$x = 1 - exp(-u) \tag{2.31}$$

from which Eq. (2.29) follows.

CHAPTER

# 3 RESULTS AND DISCUSSION

The results gained by solving the integral presented in Eq. (2.2) using the four different methods presented in Chap. 2 can be found in this chapter. The percentage deviation of the computed integral value

The results from running the code ... can be found in the GitHub folder `https://??`. [1]

## 3.1 Solving the Integral by the Gauss-Legendre Method

When solving the integral in Eq. (2.8) by Gauss-Legendre quadrature presented in Sec. 2.2 with the limits from $-4$ to 4, $\alpha = 2$, and number of mesh points $n = 40$, the value of the integral becomes

$$GLd = 0.177118 \tag{3.1}$$

The closed-form solution to the integral with infinite limits, is given in Sec. 2.1, and has a value of $5\pi^2/16^2$. The percentage deviation between the computed estimation of the integral value and the closed-form solution, is then

$$\text{percentage deviation} = \frac{0.177118 - 5\pi^2/16^2}{5\pi^2/16^2} \cdot 100\% \approx -8.1\% \tag{3.2}$$

This deviation is quite large, and since it is difficult to run with a higher number of mesh points, it shows that this method is quite inaccurate.

## 3.2 Solving the Integral by the Laguerre Method

When solving the the integral with spherical coordinates with $r_i$ being the root of the $n$'th Laguerre polynomial and $\theta_i$ and $\phi_i$ being roots of the $n$'th Legendre polynomial, the integral value with $n = 12$ is found to be

$$GLr_{12} = 0.192155 \tag{3.3}$$

which deviates from the closed-form solution by

$$\text{percentage deviation} = \frac{0.192155 - 5\pi^2/16^2}{5\pi^2/16^2} \cdot 100\% \approx -0.32\% \tag{3.4}$$

[1] FiXme Note: correct the above lines

which is a great deal smaller than the deviation when using the Gauss-Legendre method for $n = 40$.

However, if the computed Laguerre method is run for $n = 30$, the computed value for the integral is

$$GLr_{30} = 0.195069 \tag{3.5}$$

which deviates from the closed form solution by

$$\text{percentage deviation} = \frac{0.195069 - 5\pi^2/16^2}{5\pi^2/16^2} \cdot 100\% \approx 1.2\% \tag{3.6}$$

and hence the deviation from the closed-form solution is found to be larger for $n = 30$ than for $n = 12$, which is intuitively not what was to be expected.

## 3.3 Monte Carlo method

### 3.3.1 Brute force

Monte Carlo method is used on the same problem as before. It uses random numbers and a uniform probability distribution. It gives the result Tab. 3.1. Where the Integral value is close to but not exactly the same as the closed form solution. Also it looks like there is something not working correctly with the standard deviation, it gives values that are to small, when comparing the result with the closed form solution.

***Table 3.1.*** Results from the brute force method.

| Number of integration points | Integral value | standard deviation |
|---|---|---|
| $10^7$ | 0.19988 | 0.000282091 |
| $10^8$ | 0.175551 | 0.000126166 |
| $10^8$ | 0.183602 | 0.000168347 |

## 3.4 Computational Time for each Method

The Tab. 3.2 and Tab. 3.3 shows computational time for some selected runs of the Gauss-Legendre, Laguerre, brute force Monte Carlo and improved Monte Carlo methods for different $N$'s together with the gained result by running the codes with the selected $N$'s.

***Table 3.2.*** Computational time for selected runs of the Gauss-Legendre and Laguerre method.

| Method | $N$ | Result | Time (sec) |
|---|---|---|---|
| Legendre | 12 | 0.0548094 | 0 |
| | 20 | 0.127513 | 15 |
| | 30 | 0.163743 | 179 |
| Laguerre | 12 | 0.192155 | 1 |
| | 20 | 0.195636 | 29 |
| | 30 | 0.195069 | 339 |

There is a big difference between the methods, both in run time and accuracy. Both the Gaussian-Legendre and Gaussian-Laguerre methods are slow and inaccurate compared to the Monte Carlo methods. They also

***Table 3.3.*** Computational time for selected runs of the brute force Monte Carlo (MC) and improved Monte Carlo method.

| Method | $N$ | Integral | Standard deviation | Time (sec) |
|---|---|---|---|---|
| Brute force MC | $10^6$ | 0.0975576 | 0.0170543 | 0 |
| | $10^7$ | 0.229248 | 0.0444382 | 6 |
| | $10^8$ | 0.19671 | 0.0192389 | 65 |
| Improved MC | 1000 | 0.219422 | 0.033404 | 0 |
| | $10^4$ | 0.19491 | 0.0124651 | 0 |
| | $10^5$ | 0.188323 | 0.00347509 | 0 |
| | $10^6$ | 0.192977 | 0.000980611 | 0 |
| | $10^7$ | 0.192412 | 0.00031521 | 8 |
| | $10^8$ | 0.19286 | 0.000104762 | 82 |

require small N values to be able to run, which also limits their accuracy. The Gaussian-Laguerre method produces more accurate results than the Gaussian-Legendre method (see Tab. 3.2, but the computational time is almost twice as big.

Both the brute force Monte Carlo method and the improved gives more accurate results than the Gaussian methods when the N value is high. At low N values they are giving worse results than the Gaussian methods as seen in Tab. 3.2 and Tab. 3.3. But the run time is much better, they can do large N values quicker than the Gaussian. If comparing the two Monte Carlo methods listed in Tab. 3.3, the improved one gives better results with a small N than the brute force one, but the brute force method is quicker as it has fewer calculations to make. But as the improved one gives the same accuracy with a smaller N value, it can get the same accuracy just as fast or faster than the brute force one.

CHAPTER

# 4 CONCLUSION

Conclude.... conclude.... conclude....