Nummerical Integration Using Gaussian Quadrature and Monte Carlo Method

ANDREAS FAGERHEIM*

Department of Physics, University of Oslo, Norway

October 23, 2019

Abstract

This article set forth to integrate a six-dimensional integral which is used to determine the ground state correlation energy between two electrons in a helium atom. The integral appears in many quantum mechanical applications. We will first solve the integral through a brute force manner using Gauss Quadrature (Gauss-Legendre) and Monte Carlo method. By changing coordinate frame and choosing a better suited probability distribution function (PDF) we implemented improved methods respectively Gauss-Laguerre and Monte Carlo with Importance Sampling. This article will show that by doing these implementations of improvement we can get better precision off our numerical methods for integration.

I. INTRODUCTION

Numerical integration is of great importance for the scientific world. Not all integrals can be solved analytically and we must therefore find other methods for solving them. There are many aspects in implementations of numerical methods and each method comes with its drawbacks and at the same time some strengths. In this article we will set forth to look closer on four such methods. Throughout the article the concepts and methods discussed will have its foothold from Computational Physics [Hjorth-Jensen, 2015].

We will first set the framework for the physical problem we are solving and the integrals we are interested in solving. Thereafter implementation of our methods Gauss-Legendre, Gauss-Laguerre, brute force Monte Carlo and Monte Carlo with Importance sampling. Following the methods section is our results of the methods evaluating the integral. Last we will form a conclusion of our findings.

II. INTRODUCTION/THEORY

The wave function of each electron can be assumed to be modelled like the single-particle wave function of an electron in the hydrogen atom. The single-particle wave function for an electron i in the $1s$ state can the be modelled by:

$$\psi_{1s}(\mathbf{r}_i) = e^{-\alpha r_i}. \tag{1}$$

The parameter α is connected to the charge of the atom and will in our case be set to equal 2 to correspond to the helium atom $Z = 2$. Furthermore r_i is the magnitude of the position vector $\mathbf{r}_i$ and is given by

$$r_i = \sqrt{x_i^2 + y_i^2 + z_i^2}.$$

and

$$\mathbf{r}_i = x_i\mathbf{e}_x + y_i\mathbf{e}_y + z_i\mathbf{e}_z,$$

The ansatz for the wave function for two electrons is then given by the product of two so-called $1s$ wave functions as

$$\Psi(\mathbf{r}_1, \mathbf{r}_2) = e^{-\alpha(r_1+r_2)}.$$

Note that it is not possible to find a closed-form or analytical solution to Schrödinger's equation for two interacting electrons in the helium atom.

The integral we need to solve is the quantum mechanical expectation value of the correlation energy between two electrons which repel each other via the classical Coulomb interaction, namely

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

*https://github.com/AndreasFagerheim/Fys4150

Note that our wave function is not normalized, but we dont nedd to worry about this.

This integral can be solved in closed form and the answer is $5\pi^2/16^2$. This integral can be rewritten in to spherical coordinates by change of variables. We are then only left with 2 infinite integrals. The Laguerre polynomials are defined for $x \in [0, \infty)$ and we change to spherical coordinates

$$d\mathbf{r}_1 d\mathbf{r}_2 = r_1^2 dr_1 r_2^2 dr_2 dcos(\theta_1) dcos(\theta_2) d\phi_1 d\phi_2$$

want to integrate over $d\theta_i$"instead of $d\cos(\theta_i)$ and use that $d\cos(\theta_i) = sin(\theta_i) d\theta_i$ to ge

$$= r_1^2 dr_1 r_2^2 dr_2 sin(\theta_1) sin(\theta_2) d\theta_1 d\theta_2 d\phi_1 d\phi_2$$

with

$$\frac{1}{r_{12}} = \frac{1}{\sqrt{r_1^2 + r_2^2 - 2r_1 r_2 cos(\beta)}}$$

and

$$cos(\beta) = cos(\theta_1)cos(\theta_2) + sin(\theta_1)sin(\theta_2)cos(\phi_1 - \phi_2)).$$

This leads to the intgral in speherical coordinates

$$\langle \frac{1}{r_{12}} \rangle = \int d\lambda r_1^2 r_2^2 sin(\theta_1) sin(\theta_2) e^{-2\alpha(r_1 + r_2)} \frac{1}{r_{12}} \quad (3)$$

where we used $d\lambda = dr_1 dr_2 d\theta_1 \theta_2 d\phi_1 d\phi_2$. Next making the substitution $4r_1 = u''$ og $4r_2 = v''$ where we use that $\alpha = 2$ we get

$$\frac{1}{(2\alpha)^5} \int d\tilde{\lambda} \frac{sin(\theta_1) sin(\theta_2) u^2 v^2 e^{-(u+v)}}{\sqrt{u^2 + v^2 - 2uv \cdot cos(\beta)}} \quad (4)$$

where $\tilde{d\lambda} = dudvd\theta_1 \theta_2 d\phi_1 d\phi_2$

III. Methods

The integral will be solved using four numerical methods. First numerical integration method we set forth to explore is the Gaussian Quadrature which concept is to make use of a weight function $W(x)$ to give more emphasis to one part of the interval we integrate over than another. The basic idea behind this method is to approximate the integral

$$I = \int_a^b f(x) dx = \int_a^b W(x) g(x) dx \approx \sum_{i=1}^{N} w_i g(x_i) \quad (5)$$

Where w_i is the weights and obtained through orthogonal polynomials. These polynomials are orthogonal in some interval $[a, b]$ and the points x_i is constrained to lie in this interval. Different weight functions, $W(x)$ gives rise to different methods and we will look closer at Gaussian-Legendre ($W(x) = 1$) and Gaussian-Laguerre ($W(x) = x^\alpha e^{-x}$). These weight functions get their polynomials from the intervals $[-1, 1]$ and $[0, \infty\rangle$. The integral (2) we are working to solve has the limits $x \in [-\infty, \infty]$ and therefore need to rewrite the integral for to be in the right limits. By changing variable

$$t = \frac{b-a}{2} x + \frac{b+a}{2} \quad (6)$$

we can do this

$$\int_a^b f(x) dx = \frac{b-a}{2} \int_{-1}^{1} f(\frac{b-a}{2} x + \frac{b+a}{2}) \quad (7)$$

Further we have to adjust for that (5) is for only one dimension. The numerical methods will have to sum over 6 dimensions and this will be made by a number of foor-loops equal to the umber of dimensions. Under shows the idea of the implemented program used.

```
for(int i = 0;i<order;i++){
 for(int j = 0;j<order;j++){
  for(int k = 0;k<order;k++){
   for(int l = 0;l<order;l++){
    for(int m = 0;m<order;m++){
     for(int n = 0;n<order;n++){
      integral += weights[i]*weights[j]*weights[k]*weights[l]*weights[m]*weights[n]*
              int_function(roots[i],roots[j],roots[k],roots[l],roots[m],roots[n]);
                 }
```

where *int functions()* is the integral we want to evaluate and typically will bee on the form (2) and (4).

i. Gauss-Legendre Quadrature

The Gauss-Legendre method uses the weight function $W(x) = 1$ and from (5) we then want to solve the integral

$$I = \int_{-1}^{1} f(x) dx \approx \sum_{i=1}^{N} w_i g(x_i) \quad (8)$$

The integral we have (3) have limits $x \in [-\infty, \infty]$ and we can change these limits with the use of above mentioned variable change (6). By plotting the wave function for for a single particle we can narrow down our limits from $x \in [-\infty, \infty]$ to a finite interval $x \in [a, b]$. Looking at **Figure 1** we see that the single particle wave function is close to zero at $x = \pm 2$.

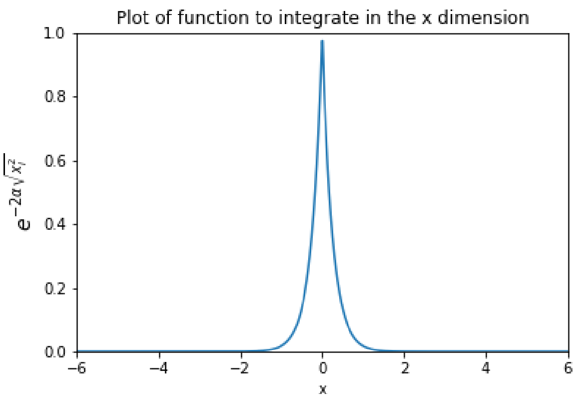

Figure 1: *Plot of the single particle wave function to find appropiate limits*

The functions *gaussLegendre (called gauleg at github library)* and *gammln* are copied from Hjort-Jensens github repository [1]. These functions returns our mesh points and weights that we want to use.

ii. Gauss-Laguerre Quadrature

The Gauss-Laguerre uses the weihgt function $W(x) = x^{\alpha} e^{-x}$. Where its associated polynomials are orthogonal in the interval $x \in [0, \infty]$ and are called Laguerre polynomials. Our rewritten integral in spherical coordinates lets us easily factor out the weight function $u^2 v^2 e^- u + v$ and from (4) the integrand becomes

$$sin(\theta_1) sin(\theta_2) \frac{1}{\sqrt{u^2 + v^2 - 2uv \cdot cos(\beta)}} \tag{9}$$

The calls below shows how we set the radial part of the integrat to be solved with Laguerre polynimials and the angular parts with Legendre polynomials.

We use the change of variable to integrate over the wanted limits $x \in [0, \infty]$

$$\tilde{x}_i = tan(\frac{\pi}{4}(1 + x_i) \tag{10}$$

and

$$\tilde{w}_i = \frac{\pi}{4} \frac{w_i}{cos^2(\frac{\pi}{2}(1 + x_i))} \tag{11}$$

[1] https://github.com/CompPhysics/ComputationalPhysics/blob/master/doc/Programs/LecturePrograms/programs/cppLibrary/lib.cpp

```
//distance
gauss_laguerre(r,weightsR,N,alpha);
//angles
gaussLegendre(theta_start,theta_stop,theta,weightsTheta,N);
gaussLegendre(phi_start,phi_stop,phi,weightsPhi,N);
```

Figure 2: *Code snippet of programs that sets upt the weights and mesh points*

where w_i andx_i are the original weights and mesh points in the interval [-1,1], while $\tilde{w}_i$ and $\tilde{x}_i$ are the new weights and mesh points in the interval $[0, \infty\rangle$.

iii. Monte Carlo brute force

The basic idea behind Monte Carlo methods is to evaluate some function in a random point given by probability distribution functions then sum all off the values and divide by number of samples made. The integral can be approximated by

$$I = \langle f \rangle = \int_0^1 f(x) p(x) dx \approx \frac{1}{N} \sum_{i=1}^{N} f(x_i) p(x_i) \tag{12}$$

The simplest form of distribution is the uniform distribution which generates numbers $x \in [0, 1]$. We therefore need our integral to go over the limits $[0, 1]$ This can be done by variable change

$$z_i = a + (b - a) x_i) \tag{13}$$

This is simply implemented in the code where x_i is generated by *ran 0* which return a random number between 0 and 1. We then calculate the integrand which will correspond to (2) and the same as used in Gauss-Legendre method.

Figure 3: *Code snippet of how the variable change occur in the program where we see x_i is a random number given by the function ran 0*

```
for(int i = 0;i<=samples;i++){
    //creates points to integrate over
    for(int j = 0;j<dim;j++){
        x[j] = a + (b-a)*ran0(&idum);
    }
    fx = int_MC_brute(x);
    int_mc +=fx;
    sum_sigma += fx*fx;
```

When we make the change of variable we

Figure 4: *Code snippet of how the variable change occur in the program where we see x_i is a random number given by the function ran 0*

```
for(int i = 0;i<=samples;i++){
    //creates points to integrate over
    for(int j = 0;j<dim;j++){
        x[j] = a + (b-a)*ran0(&idum);
    }
    fx = int_MC_brute(x);
    int_mc +=fx;
    sum_sigma += fx*fx;
```

also get the jacobideterminnant which is in this case will correspond to the volume we integrate over. The jacobi determinant is given by

$$\prod_{i=0}^{D}(a_i - b_i). \tag{14}$$

Further when using Monte Carlo methods we make use of the statistical error variance σ^2 for evaluating the precision. This tells us to which extent f deviates from its average over the integration region and is given by

$$\sigma^2 = \langle f^2 \rangle - \langle f \rangle^2 \tag{15}$$

The function *ran0* are copied from Hjort-Jensens github repository [2]. This function gives us the ability to generate random numbers in the limit $[0,1]$.

iv. Monte Carlo method improved

The idea is to make a variable change of our integrand so that we can make use of a PDF which makes the method converge faster towards an answer. By making the variable change to spherical coordinates (4) as earlier we can achieve this. The general integral transform

$$I = \int_a^b F(x)dx = \int_a^b p(x)\frac{F(x)}{p(x)}dx \approx \int_{\tilde{a}}^{\tilde{b}} p(x)\frac{F(x)}{p(x)}dx$$

$$\int_{\tilde{a}}^{\tilde{b}} p(x)\frac{F(x)}{p(x)}dx \approx \frac{1}{N}\sum_{i=1}^{N}\frac{F(x(y_i))}{p(x(y_i))} \tag{16}$$

The rewritten integral (4) makes lets us factor out e^{-u} and e^{-v} and this motivates use of a exponential probability distribution. The variable change is done by $u_i = -ln(1-x_i)$ and $u_i = -ln(1-x_i)$ which sets up the integral to go over $[0,\infty]$. While $\theta \in [0,\pi], \phi \in [0,2\pi]$.

Figure 5: *Code snippet of how the variable change with the exponential PFD occur in the program where we see x_i is a random number given by the function ran 0*

```
for(int i = 1;i<n;i++){
    for(int j = 0;j<2;j++){          //radial points
        double y = ran0(&idum);
        x[j] = -0.25*log(1.-y);      //p.367
    }for(int j = 2;j<4;j++){          //angular (theta
        x[j] = pi*ran0(&idum);
    }for(int j = 4;j<6;j++){         //angular (phi)
        x[j] = 2*pi*ran0(&idum);
    }
    fx = int_MC_spherical(x);
    int_mc +=fx;
    sum_sigma +=fx*fx;
}
```

IV. Results

i. Gauss-Legendre Quadrature

The implemented Gauss-Legendre Quadrature method yields the results given in **Table 1** when used to solve (2). With $N = 35$ the relative error is at its lowest, but is still 1.7%. For $N > 25$ this method also becomes time consuming where it for N = 40 takes 11 minutes to produce a relative error of 2.1%.

N	Integral	Relative error	Time
5	0.354602	0.8395	0.00248 s
10	0.129834	0.3265	0.156 s
15	0.199475	0.0348	1.82 s
20	0.177065	0.08145	10.6 s
25	0.18911	0.01897	38.8 s
30	0.185796	0.03616	116 s
35	0.189387	0.01753	294 s
40	0.188867	0.02125	654 s

Table 1: *Results from using Gauss-Legendre Quadrature for calculating the integral with an exact solution equal to* 0.192766

[2] https://github.com/CompPhysics/ComputationalPhysics/blob/master/doc/Programs/LecturePrograms/programs/cppLibrary/lib.cpp

ii. Gauss-Laguerre Quadrature

This result from integrating with Gauss-Laguerre Quadrature method is shown in**Table 2**. This method converges faster towards the exact value 0.192766. And with N = 25 we have a precision of 3 digits equal to the exact answer. Time used compared to Gauss-Legendre is not so different. Gauss-Laguerre can be said to outperform Gausse-Legendre on precision where it already at N = 15 has better precision than GL at any N.

N	Integral	Relative error	Time
5	0.17345	0.1002	0.00447 s
10	0.18645	0.03273	0.173 s
15	0.18975	0.0156	1.93 s
20	0.19108	0.008738	10.7 s
25	0.19174	0.005322	41.2 s
30	0.192113	0.003386	125 s
35	0.19234	0.002914	309 s
40	0.192496	0.001415	690 s

Table 2: *Results from using Gauss-Laguerre Quadrature for calculating the integral with an exact solution equal to* 0.192766

iii. Monte Carlo brute force

Integrating numerically using brute force Monte Carlo method yields the results shown in **Table 3**. Here the results fluctuates where its closer at $N = 10^5$ than for nome higher values of N. This makes little sense and makes it harder to compare the reults to the other methods.

N	Integral	Variance	Time(s)
10^5	0.19726	0.03705	0.0392 s
10^6	0.13695	0.01793	0.342 s
10^7	0.16192	0.01744	3.59 s
10^8	0.19467	0.01152	36.1 s
10^9	0.19449	0.01823	350 s

Table 3: *Results from using brute force Monte Carlo method for for calculating the integral with an exact solution equal to* 0.192766

The error in Monte Carlo methods scales with

$$error \sim \frac{1}{\sqrt{N}}. \tag{17}$$

Looking at this we see that the error should decrease as N gets higher. In our case this does not occur for the brute force Monte Carlo method implemented which leads to the result produced to bee somewhat less valid.

iv. Monte Carlo with importance sampling

The results from integration using Monte Carlo with importance sampling is shown in **Table 4**. The precision of this methods is far superior to the three others. With $N = 10^6$ its precision is already at the level of 3 leading digits and it takes only 0.4s. Gauss-Legendre do not reach this precision even for N = 30 and it takes nearly 2 minutes to reach a value vit relative error of 0.036. Gauss-Legendre reaches precision level of 3 leading digits but it takes 2 minutes and 5 seconds. The brute force Monte Carlo method is as said hard to compare to the other methods and needs a higher number for N to converge. For $N = 10^9$ the both Monte Carlo methods start to take quite some time to evaluate the integral. We see here that the error also decreases for a higher N which coincides with how the error (17) should scale.

N	Integral	Variance	Time(s)
10^5	0.19437	0.0082	0.0433 s
10^6	0.19298	0.00633	0.4 s
10^7	0.19271	0.01058	4.03 s
10^8	0.19279	0.00889	39.8 s
10^9	0.19276	0.00876	396 s

Table 4: *Results from using the method Monte Carlo with importance sampling for calculating the integral with an exact solution equal to* 0.192766

V. Conclusion

It is clear that to the Monte Carlo with importance sampling is far superior to the other methods both in time and precision. The Monte Carlo with I.S. reaches a result with 3 digits precision in 0.4 seconds compared to Gauss-Laguerre which takes 2 minutes and 5 seconds. suspect the brute force Monte Carlo method actually to converge faster than it did in this article and this may comes from some fault in my implementation. During the implementation of these methods there have not been used any unit tests. Due to the format of the research done the results from the methods them self can said to be a form of testing when we can compare them to the known exact value ($\frac{5\pi^2}{16^2}$) of the integral.

References

[Hjorth-Jensen, 2015] Hjort-Jensen, M. (2015). Computational Physics.

[Hjorth-Jensen] Hjort-Jensen, M. https://github.com/CompPhysics/ComputationalPhysics