FYS 4150-Computational Physics-Project 3

Computation of the ground state correlation energy between two electrons in a helium atom

Jeyalakshmi Thoppe Subramanian

October 2019

---

1 Abstract

This project aims at computation of the ground state correlation energy between two electrons in a helium atom numerically as a six-dimensional integral . This task is attempted and achieved with different methods, namely: numerical methods like Gauss-Legendre quadrature, Gauss-Laguerre quadrature and with statistical sampling method like Monte Carlo method (brute force Monte Carlo and monte carlo with importance sampling). The results are then compared and discussed. The code was checked for time improvement with parallelization aspect.The python code and results are is provided at final programs and final results folders at `https://github.com/jeyalakt/4150COMP_PHY/tree/master/project3`

2 Description of the problem

We approximate the wave function for two electrons in the helium atom with the product of the two single-particle hydrogen-like wave function:

$$\Psi(\mathbf{r}_1, \mathbf{r}_2) = e^{-\alpha(r_1+r_2)} \tag{1}$$

Here the normalisation factor is not considered for this project. We then want to compute the expectation value for the Coulomb interaction energy:

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

This integral can be solved analytically, resulting in $\frac{5\pi^2}{16^2}$. In the following analysis we will compare our results with this value.

Concept of multidimensional integral with integral basics[1] with Gaussian quadratures such as Legendre and Laguerre are used to compute the integral. Then the statistical simulation method Monte carlo method in brute force form and with importance sampling is used to compute the integral

2.1 Gauss Legendre quadrature

At first Gauss-Legendre quadrature[2] was applied to compute the integral. We will now use a set of orthonormal polynomials [2][3] (e.g. Legendre, Laguerre...) , defined on the interval $[a, b]$, where a, b could also be $\pm\infty$. We now want to approximate our function with a polynomial of order $2N-1$, so

$$\int_a^b f(x)dx \approx \int_a^b P_{2N-1}(x)dx = \sum_{i=1}^{N} w_i P_{2N-1}(x_i) \simeq \sum_{i=1}^{N} w_i f(x_i) = \sum_{i=1}^{N} \tilde{w}_i g(x_i) \tag{3}$$

with $f(x) = W(x)g(x)$, $\tilde{w}_i = w_i W(x_i)$ and x_i zeros of the orthogonal polynomials connected with the weight function $W(x)$.
Different integration intervals lead to different choices for the set of orthogonal polynomials and consequently to different weights $W(x_k)$ and different functions $g(x)$.

2.1.1 Legendre Polynomials

When the integration interval spans over a finite interval $[a, b]$ we use a particular set of orthogonal polynomials: Legendre Polynomials.
Legendre polynomials are defined over $[-1, 1]$ and for these polynomials the weight functions are:

$$W(x) = 1 \tag{4}$$

and the following orthogonality conditions holds for $m \neq n$:

$$\int_{-1}^{1} L_m(x)L_n(x)dx = 0 \tag{5}$$

and for $m = n$ the normality condition holds:

$$\int_{-1}^{1} \big(L_n(x)\big)^2 dx = \frac{2}{2n+1} \tag{6}$$

Finally Legendre polynomials can be defined in a recursive way:

$$\mathbf{L}_0(x) = 1,\ (j+1)\mathbf{L}_{j+1}(x) + j\mathbf{L}_{j-1}(x) - (2j+1)x\mathbf{L}_j(x) = 0. \tag{7}$$

For evaluating the integral we approximate the function $f(x) \approx P_{2N-1}(x)$ and given the orthonormality of the Legendre Polynomials we can decompose the former polynomial as

$$P_{2N-1}(x) = L_N(x)P_{N-1}(x) + Q_{N-1}(x)$$

Since the polynomial $P_{N-1}(x)$ can also be decomposed in terms of these orthonormal polynomials of degree 1, 2, ... $N-2$, $N-1$:

$$P_{N-1}(x) = \sum_{k=0}^{N-1} \alpha_k L_k(x) \tag{8}$$

we obtain:

$$\int_a^b P_{2N-1}(x)dx = \int_a^b (L_N(x)P_{N-1}(x) + Q_{N-1}(x))dx = \int_a^b Q_{N-1}(x)dx. \tag{9}$$

Moreover, when x equals one of the N roots of L_N we have $P_{2N-1}(x_k) = Q_{N-1}(x_k)$ exactly. We also note that evaluating our function $f(x)$ in these points, we obtain N independent values which fully define the polynomial Q_{N-1}. Therefore, we will choose these points as mesh points. Our aim now is to estimate our integral as

$$\int_a^b f(x)dx \approx \int_a^b Q_{N-1}(x)dx \approx \sum_{k=1}^{N} \omega(x_k)f(x_k) \tag{10}$$

for some weights $w(x_k)$. In this case the integration interval is $[a, b]$, where $a, b \in \mathbb{R}$ and it can be easily transformed into the interval $[-1, 1]$ via a simple substitution $y = -1 + \frac{2(x-a)}{b-a}$. It will be shown that such a choice leads to straight-forward calculations.

Turning back to equation (9), our following task is to develop $Q_{N-1}(x)$ in terms of Legendre polynomials: $Q_{N-1}(x) = \sum_{i=0}^{N-1} \alpha_i L_i$. Now, since $L_0(x) = 1$ we get

$$\int_{-1}^{1} Q_{N-1}(x)dx = \sum_{i=0}^{N-1} \alpha_i \int_{-1}^{1} L_0(x)L_i(x)dx = 2\alpha_0 \tag{11}$$

due to orthonormality relations.
Defining the matrix $\mathbf{L}$ as the matrix of the coefficients of Legendre polynomials and $\boldsymbol{\alpha}$ as the vector of the projection of the polynomial $\mathbf{Q}_{N-1}$ along the i-th Legendre polynomial, the following relation holds:

$$\mathbf{Q}_{N-1}(x_k) = \mathbf{L}\boldsymbol{\alpha} \tag{12}$$

hence

$$\mathbf{L}^{-1}\mathbf{Q}_{N-1}(x_k) = \boldsymbol{\alpha} \tag{13}$$

from equations (11) and (13), we get:

$$\int_{-1}^{1} f(x)dx \approx \sum_{i=0}^{N-1} \omega_i f(x_i) \tag{14}$$

where x_i are the zeros of the Legendre polynomial of degree N and the vector of the weights is

$$\tilde{w_i} = 2(\mathbf{L}^{-1})_{0i}$$

.

So, we first need to calculate the nodes and the weights and use them for integral evaluations, which greatly speeds up the calculation compared to simple integration methods as Simpons rule, rectangule rule and trapezoidal rule

Legendre polynomials are defined by the following recursive rule:

$$P_0(x) = 1$$

$$P_1(x) = x$$

$$nP_n(x) = (2n-1)xP_{n-1}x - (n-1)P_{n-2}x$$

The recursive equation for their derivative are:

$$P_n'(x) = \frac{n}{x^2-1}(xP_n(x) - P_{n-1}(x))$$

The roots of those polynomials are in general not analytically solvable, so they have to be approximated numerically, for example by Newton-Raphson iteration:

$$x_{n+1} = x(n) - \frac{f(x_n)}{f'(x_n)}$$

The first guess x_0 for the i-th root of a -order polynomial P_n can be given by

$$x_0 = \cos(\pi * \frac{i - \frac{1}{4}}{n + \frac{1}{2}})$$

After we get the nodes x_i, we compute the appropriate weights by:

$$w_i = \frac{2}{(1-x_i^2)(P_n'(x_i))^2}$$

once we have the nodes and the weights for a n-point quadrature rule, we can approximate an integral over any interval $[a, b]$ by

$$\int_a^b f(x)dx \approx \frac{b-a}{2} \sum_{i=1}^{n} w(i) f(\frac{b-a}{2}x_i + \frac{a+b}{2}) \tag{15}$$

In the given equation 2 ,

$$r_i = x_i e_x + y_i e_y + z_i e_z$$

and $r_i = \sqrt{x_i^2 + y_i^2 + z^2}$

so the component $\frac{1}{|r_1 - r_2|}$ is calculated as vector distane between in this case r_1 and r_2

The interval values of a and b are first taken as -1 and 1 and as per the hint given in the project, the single-particle wave function $e^{-\alpha(r_i)}$ is more or less zero at $r_i \approx \lambda$ the intervals are replaced with λ values in this case $[-5, 5]$.

2.2 Gauss Laguarre Quadrature

The Legendre polynomials [2] [3] are defined for x $\in [-1, 1]$ and The gauss legendre quadrature gave a ad hoc procedure. The results can be improved by changing the coordinate system and apply the Laguerre polynomials. The Laguerre polynomials are defined for x $\in [0, \infty]$ and if we change to spherical coordinates

$$dr_1 dr_2 = r_1^2 dr_1 r_2^2 dr_2 d\cos(\theta_1) d\cos(\theta_2) d\phi_1 d\phi_2 \tag{16}$$

with

$$\frac{1}{|r1 - r2|} = \frac{1}{\sqrt{r_1^2 + r_2^2 - 2r_1 r_2 \cos(\beta)}} \tag{17}$$

where

$$\cos(\beta) = cos(\theta_1)cos(\theta_2) + \sin(\theta_1)\sin(\theta_2)cos(\phi_1 - \phi_2) \tag{18}$$

our integral is now given by

$$\int_0^\infty r_1^2 dr_1 \int_0^\infty r_2^2 dr_2 \int_0^\pi d\cos\theta_1 \int_0^\pi d\cos\theta_2 \int_0^{2\pi} d\phi_1 \int_0^{2\pi} d\phi_2 \frac{exp{-2\alpha(r_1 + r_2)}}{r_{12}} \tag{19}$$

For the angles we need to perform the integrations over $\theta_i \in [0, \pi]$ and $\phi_i \in [0, 2\pi]$. For the radial part we use Gauss-Laguerre taking properly care of the integrands involving the $r_i^2 \exp -2\alpha r_i$ terms.

The following section explains the Gauss Laguerre quadrature in detail

In Gauss Laguerre polynomial the weight function in eqn 1 absorbs the exponential function $e(\alpha x)$ we rewrite eqn 1 as

$$\int_0^\infty f(x)dx = \int_0^\infty W(x)g(x)dx \tag{20}$$

2.2.1 Laguerre polynomials

When the integration interval spans over an infinite interval, we need a different set of orthogonal polynomials.

Laguerre polynomials are defined over $[0, \infty]$ and for these polynomials the weight functions are:

$$W(x) = x^\alpha e^{-x} \tag{21}$$

and the following orthogonality conditions holds for $m \neq n$:

$$\int_0^\infty x^\alpha e^{-x} G_m^\alpha(x) G_n^\alpha(x) dx = 0 \tag{22}$$

and for $m = n$ the normality condition holds:

$$\int_0^\infty x^\alpha e^{-x} (G_n^\alpha(x))^2 dx = \frac{(n + \alpha)!}{n!} \tag{23}$$

As before we want to find the weights $\tilde{w}_i$ of equation (3). Then

$$\int_0^\infty f(x)dx = \int_0^\infty W(x)g(x)dx \approx \int_0^\infty W(x)P_{2N-1}(x)dx = \int_0^\infty W(x)\Big(G_N^\alpha(x)P_{N-1}(x) + Q_{N-1}(x)\Big)dx \tag{24}$$

now the first factor of the former integral:

$$\int_0^\infty W(x)G_N^\alpha(x)P_{N-1}(x)dx = \int_0^\infty x^\alpha e^{-x} G_N^\alpha(x)P_{N-1}(x)dx = 0 \tag{25}$$

for the normalization condition of Laguerre polynomials. Now if we decompose Q with these polynomials: $Q_{N-1}(x) = \sum_{k=1}^{N-1} \beta_k G_k^\alpha(x)$ then we get

$$\int_0^\infty f(x)dx \approx \int_0^\infty W(x)Q_{N-1}(x)dx = \sum_{k=1}^{N-1} \beta_k \int_0^\infty x^\alpha e^{-x} G_k^\alpha(x) G_0^\alpha(x) = \alpha!\beta_0 \tag{26}$$

Defining the matrix $\mathbf{G}$ as the matrix of the coefficients of Legendre polynomials and $\boldsymbol{\beta}$ as the vector of the projection of the polynomial $\mathbf{Q}_{N-1}$ along the i-th Laguerre polynomial, the following relation holds:

$$\mathbf{Q}_{N-1}(x_k) = \mathbf{G}\boldsymbol{\beta} \tag{27}$$

hence

$$\mathbf{G}^{-1}\mathbf{Q}_{N-1}(x_k) = \boldsymbol{\beta} \tag{28}$$

finally, exploiting both equation (20) and (22), we get:

$$\int_0^\infty f(x)dx \approx \sum_{i=0}^{N-1} \tilde{w}_i f(x_i) \tag{29}$$

where x_i are the zeros of the Laguerre polynomial of degree N and the vector of the weights is

$$\tilde{w}_i = \alpha!(\mathbf{G}^{-1})_{0i} \tag{30}$$

2.3 Monte Carlo method

Monte Carlo method is a statistical simulation method which is widely used to calculate multidimensional integral. Statistical simulation methods may be contrasted to conventional numerical discretization methods, which are typically applied to ordinary or partial differential equations that describe some underlying physical or mathematical system. In many applications of Monte Carlo, the physical process is simulated directly, only requirement is that the physical (or mathematical) system be described by probability distribution functions (PDF's). Once the PDF's are known, the Monte Carlo simulation [4] can proceed by random sampling from the PDF's.

In its simplest form, i.e. when the integration interval is $[0, 1]$ and a brute force Monte Carlo method is applied, it consists in computing

$$\langle f \rangle = \frac{1}{N} \sum_{i=1}^{N} f(x_1)p(x_i) \tag{31}$$

where $p(x_i)$ is the uniform distribution and the points x_i are random generated via functions as `random.uniform()`... This method can be easily applied also to integrals which domain is in the form: $[a, b]$ via a change of variable. Assuming that $f(x_i)$ are independent, i.e. that our evaluating points x_i are really randomly chosen, the variance reads:

$$\sigma_N^2 \approx \frac{1}{N}(\langle f^2 \rangle - \langle f \rangle^2) = \frac{\sigma_f^2}{N} \tag{32}$$

hence $\sigma_N \approx \frac{1}{\sqrt{N}}$. Monte Carlo method is faster for higher dimension integrals.

Improved Monte Carlo Method Monte Carlo method can be improved with a slightly better approach: instead of picking random points in the integration multidimensional interval, it is advisable to collect more data where the value of the function value is higher. To achieve that, let $p(y)$ be a PDF whose profile resembles the function to be integrated (apart fron a scaling factor). The normalisation condition is

$$I = \int_a^b f(y)dy = \int_a^b p(y)\frac{f(y)}{p(y)}dy \tag{33}$$

which resembles our given task of evaluation of the energy for a quantum mechanical system And, performing a change of variable x $\rightarrow$ y (since random numbers are generated $[0, 1]$:

$$x(y) = \int_a^y p(y')dy' \tag{34}$$

we obtain

$$I = \int_0^1 \frac{f(y(x))}{p(y(x))}dx \tag{35}$$

Now we do plain Monte Carlo integration on the new function $\frac{f(y(x))}{p(y(x))}$, i.e to pick random points around the integration domain.

exponential distribution if p(y) = exp(−y), which is the exponential distribution, like the problem of energy state under consideration and p(x) is given by the uniform distribution with x ∈ [0,1], and with the assumption that the probability is conserved we have

$$p(y)\,dy = \exp(-y)\,dy = \,dx, \tag{36}$$

which yields after integration

$$x(y) = P(y) = \int_0^y \exp -y'dy' = 1 - \exp -y \tag{37}$$

or

$$y(x) = -\ln(1-x). \tag{38}$$

This gives us the new random variable y in the domain $y \in [0, \infty)$ determined through the random variable x ∈ [0,1] generated by functions like random(). This means that if we can factor out exp(−y) from an integrand we may have

$$I = \int_0^\infty F(y)dy = \int_0^\infty \exp(-y)G(y)\,dy \tag{39}$$

which can be rewritten as

$$\int_0^\infty \exp(-y)G(y)\,dy = \int_0^1 G(y(x))\,dx \approx \frac{1}{N}\sum_1^N G(y(x_i)) \tag{40}$$

wherex_i is a random number in the interval [0,1].

3 Discussion on Methods and Results

Gaussian Legendre quadrature The given problem was attempted with gaussian Legendre quadrature, using Legendre polynomials. In cartesian coordinates we have an integral with symmetric integration points, as the domain of Legendre polynomials, which are defined in $[-1, 1]$. Since integration domain goes from $-\infty$ to $+\infty$, the integral in cartesian coordinates reads:

$$\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} e^{-4(\sqrt{x_1^2+y_1^2+z_1^2}+\sqrt{x_2^2+y_2^2+z_2^2})}\frac{dx_1dx_2dy_1dy_2dz_1dz_2}{\sqrt{(x_1-x_2)^2+(y_1-y_2)^2+(z_1-z_2)^2}} \tag{41}$$

Since the function is of format $e^{-4(r_1+r_2)}$, the function is more or less zero after certain point, so the domain where the integral gets significant contributes is limited. So by simple calculation the integral limit are chosen close to the lamda values (eigen values) of the function where the function is closer to zero. Integral points of$[-5, 5]$ were chosen after simple calculation with different lamda values. The code also tests the points with the denominator smaller than 10^{-10} inside the region
The function was also run with different integration points form 10 to 25 . So Python code was developed using the function [3] `gauleg` defined in the library `lib.h` we have computed the weight function and we have estimated the integral as $I = \sum_{k=1}^{N} w_k[x_k]f[x_k]$. The results are provided below at table of Figure 1:

As we can see this method is not so satisfactory and the value of integral can vary a lot changing the number of integration points in an unpredictable way within given integral limits.Also Adding more points actually doesn't seem to improve the result. Execution time for this program is significant as it requires a lot of calculations in particularly 6 nested loop, which means that we have to do, with N the number of integration points, N^6 iterations and each iteration involves around 40 operations.
So it can be said that this method is not suitable for multidimensional integrals.

Laguerre Next Gaussian quadrature with a mix of Legendre and Laguerre polynomials were used. Our integral is now given by

$$\int_0^\infty r_1^2dr_1\int_0^\infty r_2^2dr_2\int_0^\pi d\cos\theta_1\int_0^\pi d\cos\theta_2\int_0^{2\pi}d\phi_1\int_0^{2\pi}d\phi_2\frac{exp{-2\alpha(r_1+r_2)}}{r_{12}} \tag{42}$$

with $cos(\alpha) = cos(\theta_1)cos(\theta_2) + sin(\theta_1)sin(\theta_2)cos(\phi_1 - \phi_2)$. Being the Laguerre polynomial defined in the interval $[0, +\infty]$. They are used for the variables r_1 and r_2 and w the Legendre polynomials are used for the angles.

This method converges better towards the exact result and result accurate to the second leading digit was obtained with $N = 20$ onwards. It was observed that as the value of denominator in the function is set too low , the function value gave unrealistic result. So this method is more dependant on the epsilon value which is used for comparison. This method seemed to be more stable than the previous one as it is more consistent . So even if this method is not so precise at least it seems to be more trustworthy.

Both Gauss Legendre and Lagaurre methods were tried with parallelisation using package 'Numba' from python because without parallelisation it was not possible to get results for n>15 in reasonble amount of time. when using Jupyter and Anaconda , there were issues when parallelisation was attempted with Process from 'multiprocessing ' package in Python . But it worked with PyCharm for N<100 . But for values of N>100 it did not work. But jit from 'Numba'was worked on both so it was used and it can be seen that both methods consumed almost similar time with parallelisation

To sum up and compare the different results of the two algorithm we show a table of the computed values and plots below:

N	a	b	exact_result	Result		Relative error	
				Legrendre	Lagaurre	Legrendre	Lagaurre
10	-5	5	0.19257	0.011165	0.177081	0.942023	0.081368
15	-5	5	0.19257	0.236879	0.187371	0.230093	0.027987
20	-5	5	0.19257	0.096789	0.194786	0.497385	0.010478
25	-5	5	0.19257	0.219647	0.193423	0.140608	0.003411

Figure 1: Gaussian Methods Result

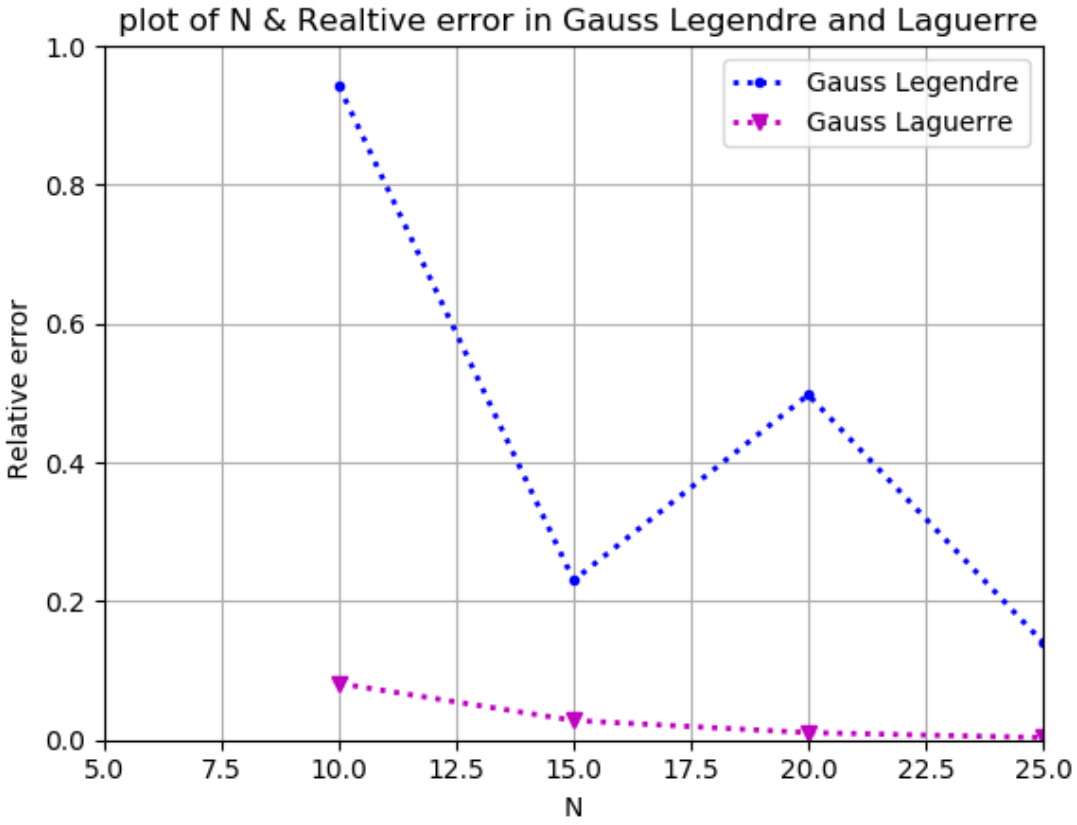

Figure 2: Gaussian methods- N vs Relative error

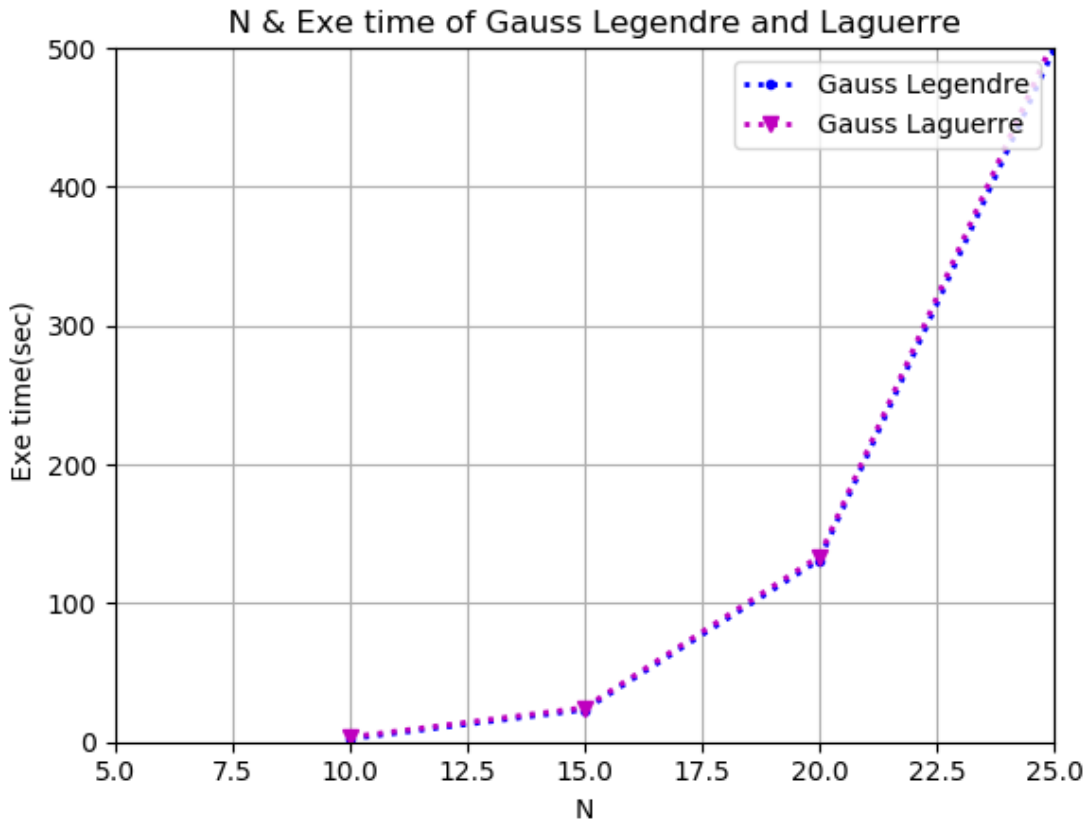

Figure 3: Gaussian methods- N vs exe time with Parallel processing

Monte Carlo For the final method , Monte carlo simulation in brute force approach and an improved version were used . In The first- Monte carlo brute force method , the uniform distribution was used to get random points over a fixed interval;The limits were chosen as the lamda(eigen) values as in the same way for the Gaussian-Legendre quadrature, namely $(-5, 5)$.
For second improvised method , the integral was transformed to spherical coordinates and a change of variables for the radial coordinate to make it cover all the range from 0 to infinity.The mapping function $y(x) = -ln(1-x)$ was used.
Importance sampling was used with the function $p(x) = e^{-4x}$ as the new probability function which simplified a lot integrand function as well, where x is a number generated by a random() function.The integrand function was multiplied with the proper jacobian value. Hence in this improved method,it has been possible to sample it mainly where the given function values were bigger (where the importance sampling function value were bigger), instead of collecting a lot of points in regions where it is basically 0.

The results and plots are provided below:

N	a	b		Result		
			exact_result	MC brute	MC _improved	MC-imp_paralle
100000	-5	5	0.19257	0.034806	0.194124	0.191812
500000	-5	5	0.19257	0.104994	0.193304	0.192351
1000000	-5	5	0.19257	0.083068	0.191073	0.191032
5000000	-5	5	0.19257	0.195673	0.192748	0.192595
10000000	-5	5	0.19257	0.148979	0.193341	0.193209

Figure 4: Monte Carlo methods- Exact Result vs Calulated Results

N	a	b	Relative error		
			MC brute	MC _improved	MC-imp_paralle
100000	-5	5	0.819255	0.008068	0.00394
500000	-5	5	0.454776	0.003812	0.00114
1000000	-5	5	0.568637	0.007777	0.007986
5000000	-5	5	0.01611	0.000922	0.000129
10000000	-5	5	0.226367	0.004	0.003314

Figure 5: Monte Carlo methods- N vs relative error

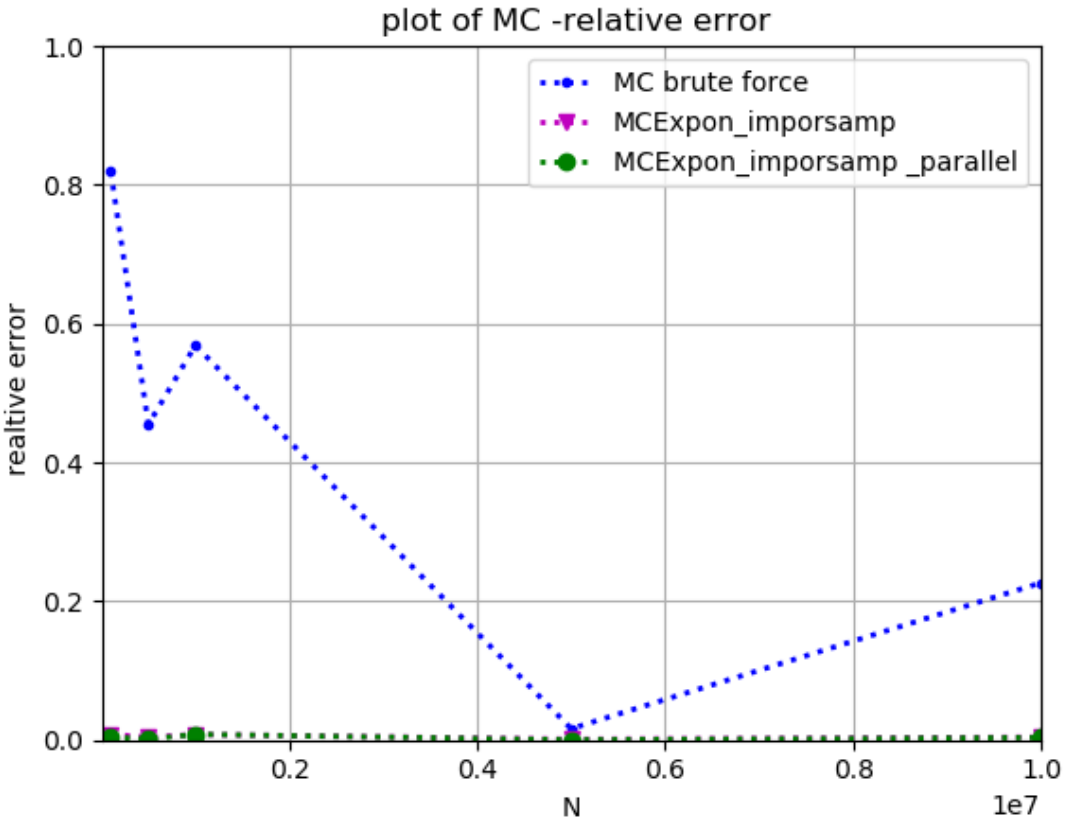

Figure 6: Monte Carlo methods- N vs relative error

N	a	b	Sigma		
			MC brute	MC _improved	MC-imp_paralle
100000	-5	5	1.21E-09	0.054999	0.001363
500000	-5	5	1.10E-08	0.043947	0.000978
1000000	-5	5	6.90E-09	0.038472	0.001493
5000000	-5	5	3.83E-08	0.045483	0.001439
10000000	-5	5	2.22E-08	0.051185	0.0001

Figure 7: Monte Carlo methods- N vs Sigma Value

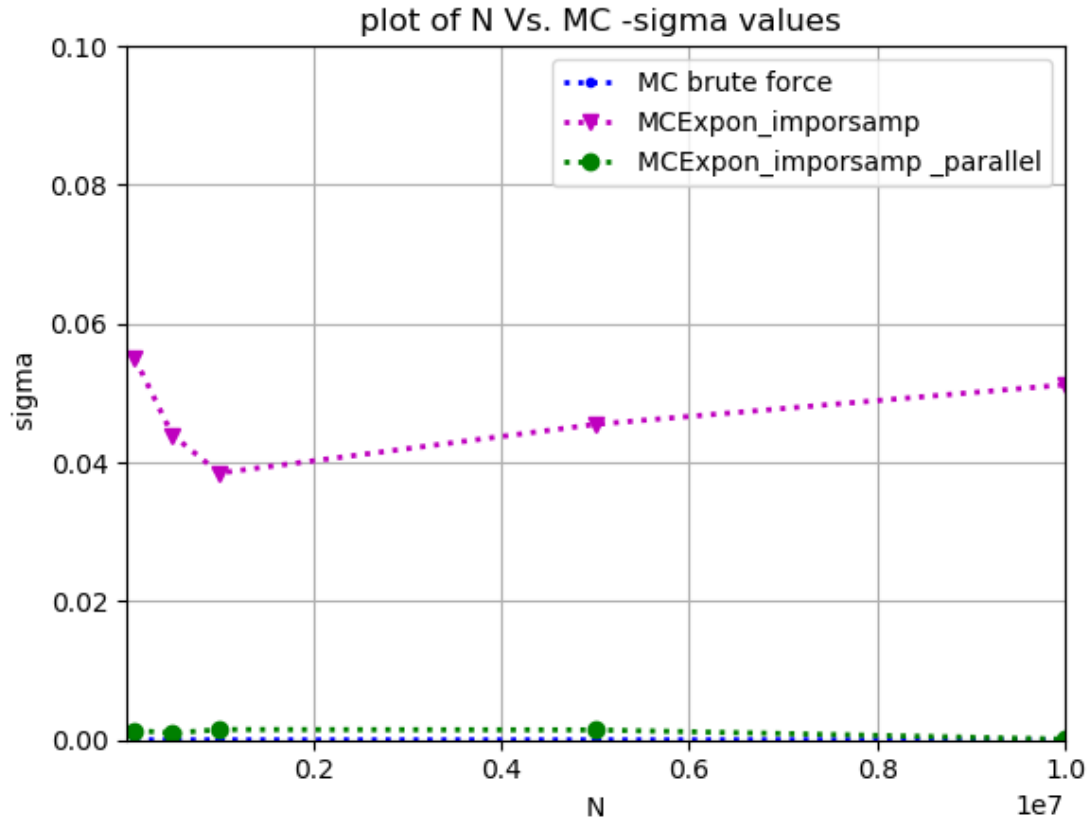

Figure 8: Monte Carlo methods- N vs Sigma Value

N	a	b	Execution Time(sec)		
			MC brute	MC _improved	MC-imp_paralle
100000	-5	5	1.173858	1.018276	0.465753
500000	-5	5	6.03315	5.086325	1.81215
1000000	-5	5	11.632929	10.56426	3.633324
5000000	-5	5	59.339541	51.738686	18.196394
10000000	-5	5	116.214675	103.845157	37.911287

Figure 9: Monte Carlo methods- N vs Exe time

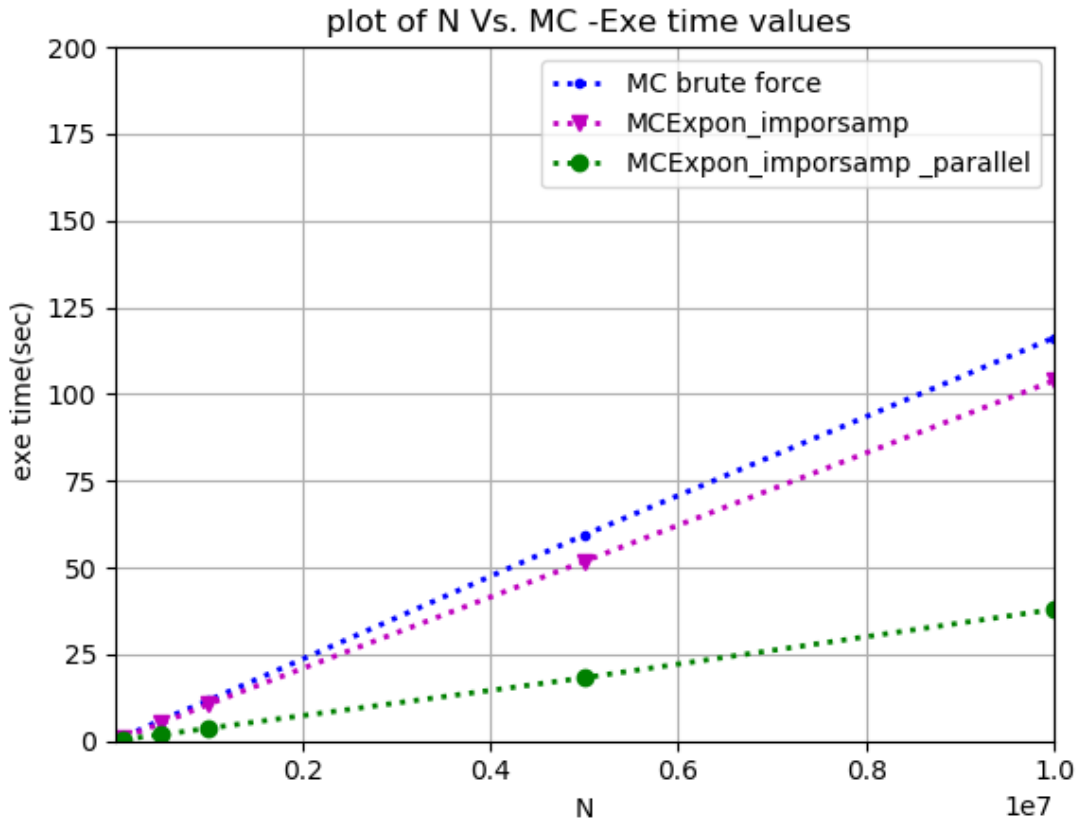

Figure 10: Monte Carlo methods- N vs Exe time

As it is observed the variance decreases as we increase the number of integration points, as expected. However the brute force integral requires a lot of points for its value to get close to the analytical one. The method with important sampling, is almost exact for all the choices of number of integration points , with significantly lower σ. The importance sampling allows to compute the integral with really few points much less then the Gaussian quadrature and the brute force Monte Carlo.

Parallelization The last part of the project was to implement parallelization in the code . The 'Numba' library was used for this purpose.With minimum trivial changes to the code considerable speed up was achieved as shown. When tried to use the POOL, PROCESS and THREAD from multiprocessing package in Python in Jupyter notebook , unreasonable hangs were observed and hence switched over to PYcharm.Multiprocessing worked in PYcharm environment for N <100. But the package Numba worked well in both environments without any problem

As we can see execution times are considerably decreased after the parallelization. This could be due to the

fact that we set up the parallel calculation in a way that avoids data races.It could not be confirmed if the time of execution is linearly correlated with the number of cores
This time analysis however showed us that improved Monte carlo method with importance sampling is the fastest with more reliable than the brute force Monte carlo method.

4 Conclusions

Among numerical representations, gaussian quadrature with Legendre polynomials resulted to be the less reliable method, since it depends on the number of integration points. Laguerre polynomials gave much more precise and consistant results
Monte Carlo methods are faster but brute force Monte Carlo is less accurate with limited integration intervals. Finally, a change of coordinates and the implementation of importance sampling with exponential integration increased the precision of our results and with parallelisation results were achieved in quicker time.

5 Acknowledgement

I would like to thank all the Lab instructors in this course who answered all queries very patiently and clarified all doubts.

6 Critique

It can be attmepted to present video lectures where huge derivations and calculations are involved . it can help us to rerun the video and understand them better at our own pace.
The literature and references for a particular project are scattered throughout lecture notes and slides which can be a challenge to search . If these are presented along with the project description it can be easier
From previous experience of lab guidance,common doubts can be cleared for all participants which will save lot of time and energy for the Instructors and give the participants reasonable waiting time.

References

[1] C.Henry Edwards, David E,Penny. *Calculus, Early Transdentals- parts of Chapter 5 , 13*. Pearon, 2007.

[2] Morten Hjorth-Jensen:,
`http://compphysics.github.io/ComputationalPhysics/doc/pub/integrate/html/integrate-reveal.html`

[3] Morten Hjorth-Jensen:,
`https://github.com/CompPhysics/ComputationalPhysics/tree/master/doc/Projects/2019/Project3/CodeExamples`

[4] Morten Hjorth-Jensen:,
`http://compphysics.github.io/ComputationalPhysics/doc/pub/mcint/html/mcint-reveal.html`