FYS3150: Project 3: Numerical Integration

Henrik Haugerud Carlsen, Martin Moen Carstensen and Øyvind Augdal Fløvig

October 22, 2019

Abstract

In this report we have take a look at Gauss-Legendre and Gauss-Laguerre Quadrature as well as Monte Carlo integration for solving integrals. The problem more specifically is to determine the expectation value of the correlation energy between two electrons by solving an integral. The time spent for the two Gaussian quadrature method was pretty similar only that Laguerre converted faster. Here for N = 21 we got the integral value of $I = 0.1948$ while the Legendre gave $I = 0.2043$ for similar N. Significant speedup was achieved for Monte Carlo for sufficiently large values of N. The exponential approach for Monte Carlo integration was faster than the uniform distribution approach and converged on the correct value of the integral faster. $N = 1000000$ resulted in 0.196 for the exponential Monte Carlo approach while the uniform distribution approach resulted in $I = 0.213$ for the same value.

1 Introduction

This report is going to look at different ways of integrating the same integral which is the expectation value of the Coulomb interaction. This will be done by using Gaussian quadrature method with Legendre polynomials for the Cartesian coordinates and a mix of both Legendre and Laguerre polynomials for when we switch to spherical coordinates. Furthermore we want to use Monte Carlo Integration as well to calculate the integral. This is done so the different methods can be compered in terms of speed, accuracy. This is done to see how different integration methods interact differently and what the benefits of using them are. Lastly parallelization of the Monte Carlo method will be looked at for, potentially, even greater time reduction. Parallelization is an important aspect of scientific programming as it allows for large number of computations to be carried out much faster, and it is for that reason crucial for up and coming science students to have some knowledge of the concept.

2 Method

This report is going to look at different numerical methods of integrating the same integral. We are going to consider the six-dimensional integral which is used to determine the ground state correlation energy between to electrons in a helium atom. The wave function of one electron in the 1s state with position

$$\vec{r}_i = x_i\hat{e}_x + y_i\hat{e}_y + z_i\hat{e}_z$$

is

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

Here α is a parameter which we are going to fix to $\alpha = 2$ to model the helium atom of charge $Z = 2$ and

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

The anzats of two electron is then given as the product of two single electron 1s states which is

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

It is not possible to find a closed-form or analytical solution of this Schrodinger's equation for two interacting electrons in the Helium atom. Instead we can find the quantum mechanical expectation value of the correlation energy between the two electrons. This can be done with the equation

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

2.1 Gaussian Quadrature

Any integral of the form

$$I = \int f(x)dx = \int W(x)g(x)dx$$

can be written as a sum of the wieghts and function values at the form

$$I = \int f(x)dx = \sum w_i g(x_i)$$

Here the weights and mesh points are found using orthogonal polynomials of degree N. And these points can determine the integral value of a polynomial of degree 2N-1 exact. This is being done with two different polynomials of Legendre and Laguerre. Legendre polynomials are defined on the interval from -1 to 1 so this approximation works well on integrals with finite limits, while the Laguerre polynomials are determined on the interval from 0 to ∞ which is useful when considering sphercial coordinates

Both the integrals are being done in the program called 'Gauss_legendre.py'. Here the integration value have been found for mesh-points ranging from $N = 3$ to $N = 25$ and all the odd numbers in between. Odd numbers were chosen such that we get the mesh point of $x_i = 0$ from the legendre roots

2.1.1 Gauss-Legendre

With the Gauss-Legendre method we first find the N roots of the Legendre polynomial which are going to be the x_i mesh points. From these mesh points we can find the corresponding weights with the recurrsion formula where

$$L_{j+1}(x_i) = ((2j+1)x_i L_j(x_i) - jL_{j-1}(x_i))/(j+1)$$

From this we can find the derivated of the Nth polynomial with the formula

$$L'_N(x_i) = N\frac{(x_i L_N(x_i) - L_{N-1})}{x_i^2 - 1}$$

Now the weights are found by using

$$\omega_i = \frac{2}{(1-x_i^2)L'_n(x_i)^2}$$

From this we can now integrate any function defined on the interval from -1 to 1 with the weights ω_i and mesh points x_i. However if we want to do this for a more general integral that goes from a starting point a to an end point b we need to do a change of variables.

$$I = \int_a^b f(t)dt = \frac{b-a}{2}\int_0^1 f\left(\frac{(b-a)x}{2} + \frac{(b+a)}{2}\right)dx$$

so now we have everything we need to compute a general integral as

$$I = \int_a^b f(t)dt \approx \frac{b-a}{2}\sum_i^N \omega_i f\left(\frac{(b-a)x_i}{2} + \frac{(b+a)}{2}\right)$$

By looking at our original integral in Cartesian coordinates we get the form

$$\int e^{-2\alpha(\sqrt{x_1^2+y_1^2+z_1^2}+\sqrt{x_2^2+y_2^2+z_2^2})}\frac{dx_1dx_2dy_1dy_2dz_1dz_2}{\sqrt{(x_1-x_2)^2+(y_1-y_2)^2+(z_1-z_2)^2}}$$

By then setting the limits at $b = -a = \lambda$, where $\lambda = 3$ we can use a six dimensional Gauss-Legendre integral to calculate our original integral

2.1.2 Gauss-Laguerre

Similarly to the Gauss-Legendre method we find the new mesh points by the roots of the N-th Laguerre polynomial. Furthermore the weights are found with the same Laguerre polynomial which can be found with the recursion formula of

$$\mathcal{L}_{j+1}(x_i) = ((2j+1-x_i)\mathcal{L}_j(x_i) - j\mathcal{L}_{j-1}(x_i))/(j+1)$$

The weights of the N-th Laguerre polynomials are now given as

$$\omega_i = \frac{x_i}{(N+1)^2\mathcal{L}_{N+1}(x_i)^2}$$

So now we can calculate an integral of the form

$$I = \int_0^\infty e^{-x} f(x) dx$$

where e^{-x} is the weight function as a sum of

$$\sum_i \omega_i f(x_i) dx$$

By now changing the integral to spherical coordinates we get for

$$d\vec{r}_1 d\vec{r}_2 = r_1^2 r_2^2 sin\theta_1 sin\theta_1 dr_1 dr_2 d\theta_1 d\theta_2 d\phi_1 d\phi_2$$

with

$$\frac{1}{|\vec{r}_1 - \vec{r}_2|} = \frac{1}{\sqrt{r_1^2 + r_2^2 - 2r_1 r_2 cos(\beta)}}$$

where

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

This gives that the total integral becomes

$$I = \int_0^\infty \int_0^\infty \int_0^\pi \int_0^\pi \int_0^{2\pi} \int_0^{2\pi} e^{-2\alpha(r_1+r_2)} \frac{r_1^2 r_2^2 sin\theta_1 sin\theta_1 dr_1 dr_2 d\theta_1 d\theta_2 d\phi_1 d\phi_2}{\sqrt{r_1^2 + r_2^2 - 2r_1 r_2 cos(\beta)}}$$

So here we can use Gauss-Legendre method for integrating the angles and Gauss-Legendre for integrating the radial part. Here one of the exponential powers are used as the weight function as $W(x) = e^{-(r_1+r_2)}$

2.2 Monte Carlo Integration

A very different approach to evaluating definite integrals is what's called Monte Carlo integration. The idea is essentially to rewrite our integral as an expectation value of a function relative to some probability distribution $P(x)$. Imagine that you want to compute some integral

$$I = \int_a^b f(x)dx.$$

One way to go about computing this integral is to pick a random function value $f(x_i)$ and multiply it with the width of the interval $b - a$. This will then give you the area of some rectangle, which will likely be a very poor approximation of your integral. However, if you pick enough random points x_i and compute enough function values $f(x_i)$ and take the average of the area of the resulting rectangles, they will approach the value of the actual integral I. This is the essential idea of Monte Carlo integration. The question facing us this this article is which probability distribution to pick when choosing the random points x_i. However, analyzing the above scenario with a bit more mathematical detail we obtain the expression

$$E[f(x)] = \int_a^b f(x)P(x)dx = \int_a^b f(x)\frac{1}{b-a}dx$$

so in order to obtain to expression for the integral we need to do some additional mathematical manipulation after having computed the expectation value. In the case of the uniform distribution this only consists of multiplying by the length of the interval we want to find the integral over.

2.2.1 Brute force approach

We now have to decide upon a probability distribution $P(x)$ from which to generate the points x_i. If we don't know anything about what the function $f(x)$ looks like we might as well pick values distributed according to the most simple probability distribution there is, namely the uniform distribution, given by the equation

$$P(x) = \frac{1}{b-a}.$$

The expectation value of f then becomes

$$E[f(x)] = \int_a^b f(x)P(x) = \int_a^b f(x)\frac{1}{b-a}dx$$

which can rewritten to isolate the integral on one side

$$E[f(x)](b-a) = \int_b^a f(x)dx.$$

The expectation value can be estimated by taking the average of f for many values of x_i where x_i is sampled from the uniform distribution. This gives us the following equation for the integral

$$I = \frac{b-a}{N}\sum_{i=1}^{N} f(x_i).$$

where N is the number of x_i we evalueate the function for. When we want to compute the integral of a function with many variables we basically repeat this procedure for each variable.

2.2.2 Exponential distribution approach

From the procedure in section 2.1.2 we know that the integral can be written

$$I = \int_0^\infty \int_0^\infty \int_0^\pi \int_0^\pi \int_0^{2\pi} \int_0^{2\pi} e^{-2\alpha(r_1+r_2)} \frac{r_1^2 r_2^2 sin\theta_1 sin\theta_1 dr_1 dr_2 d\theta_1 d\theta_2 d\phi_1 d\phi_2}{\sqrt{r_1^2 + r_2^2 - 2r_1 r_2 cos(\beta)}}$$

.

This is integral has two terms, r_1 and r_2, which have an exponential form. If we picked random y values along these dimensions, the probability of finding the function at a particular height would decline exponentially. This makes these variables ideal for using the exponential distribution when doing Monte Carlo distribution. Both variables have coefficients -2α with $\alpha = 2$ which gives us coefficients of -4. The exponential distribution is give by the equation

$$P(x) = \lambda e^{-\lambda x}.$$

Picking $\lambda = 4$ we see that it fits quite nicely with the expression for the radial components in the spherical expression for the integral. We can then use the exponential distribution to rewrite our integral.

$$I = \int f(x)dx = \int \frac{f(x)}{P(x)} P(x)dx = \int g(x)P(x)dx$$

where we have made the substitution $\frac{f(x)}{P(x)} = g(x)$. We can then compute the integral by evaluating the expectation value of this new function $g(x)$, i.e.

$$I = E[g(x)] = \frac{1}{N}\sum_{i=1}^{N} g(x_i)$$

Since the function $g(x)$ is just the original function $f(x) = \frac{e^{-4x}}{A(x)}$, where $A(x)$ is the denominator, divided by the exponential distribution, we can also

compute the integral by taking the expectation value of $f(x)$ and dividing by 4,

$$I = \frac{1}{4}E[f(x)] = \frac{1}{4N}\sum_{i=1}^{N} f(x_i)$$

The exponential distribution is useful for computing the variables along the r_1 and r_2 dimensions of the integral because their values take off exponentially with time. The integral has to be written in spherical coordinates instead of cartesian when applying the exponential distribution because the exponential distribution is not well-defined for negative values of x. The variables θ_1,θ_2,ϕ_1 and ϕ_2 do not decline in an exponential fashion, and so we use the normal distribution described in the previous section when integrating these variables.

2.2.3 Variance

One metric which is useful when doing Monte Carlo simulations is the variance, defined by the equation

$$V = \frac{1}{N}\sum_{i=1}^{N}(f(x_i) - \mu)^2.$$

It measures how much the function value $f(x_i)$ deviates from the mean μ value of the function.

2.3 parallelization

Parallelization is a powerful tool to make a programs run faster. The idea is to distribute the work/computations in a program among available processors. Instead of one processor running through the program sequentially parts of the program now gets computed simultaneously saving time. The aim would be to distribute the computations among the available cores, do the computations, and then gather the computed values.To parallelize "SmarterMonteCarlo" a function "main()" containing the following is used:

```
pool = cpucount()                          ▷ "pool" counts the available cpus.
result = pool.map(MonteCarlo, list)        ▷ pool.map distributes the
computations in the function "MonteCarlo" as function of "list".
```

The idea is to find out how many cpus are available, this is done by the function "cpucount()". The parallelization of the function "MonteCarlo" is initiated in the variable "result" where the "map"-function distributes the computations done in "MonteCarlo" for all the elements in the list "Nlist".

We have parallelized the Monte Carlo integration method which uses the exponential distribution. We have done this by taking the number of iterations N and creating a new list of length $\sqrt{N}$ with elements of value $\sqrt{N}$. We then compute the Monte Carlo integral $\sqrt{N}$ times, each with $\sqrt{N}$ iterations and distributing this task over the 4 cores. We then take the mean of the resulting integral values and take this as our Monte Carlo integral.

3 Results

To begin with the Gaussian quadrature methods were tested by plotting the integral value we got for different N polynomials.

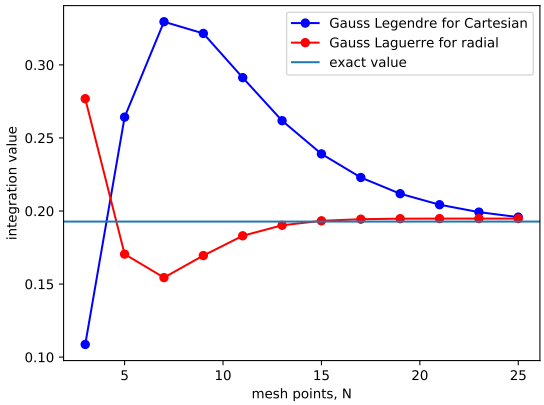


Figure 1: Legendre and Laguerre methods as a function of meshpoints

Also found the value of the integral together with the time spent to get this value. The findings is then shown for Gauss Legendre in table 1 and the Gauss Laguerre results are in table 2.

Table 1: Integration value and time spent for different mesh points with the Gaussian Legendre method

N	Integral value	time spent(s)
3	0.108672	0.0209
5	0.264249	0.4997
7	0.329525	3.0389
9	0.321518	12.4357
11	0.291261	47.9585
13	0.261821	116.628
15	0.239088	265.02
17	0.222933	574.50
19	0.211832	1120.19
21	0.204307	1898.89
23	0.199232	2654.37
25	0.195817	4542.21

Table 2: Integration value and time spent for different mesh points with the Gaussian Laguerre method

N	Integral value	time spent(s)
3	0.276857	0.01396
5	0.170492	0.3132
7	0.154422	2.3425
9	0.169505	10.4558
11	0.183022	35.6265
13	0.190218	95.9022
15	0.193285	235.989
17	0.194396	512.491
19	0.194732	962.119
21	0.194807	1684.94
23	0.194812	3203.78
25	0.194804	4877.90

The results different Monte Carlo methods are given below for different numbers of randomly selected points, N.

Table 3: Results for Standard Monte Carlo integration.

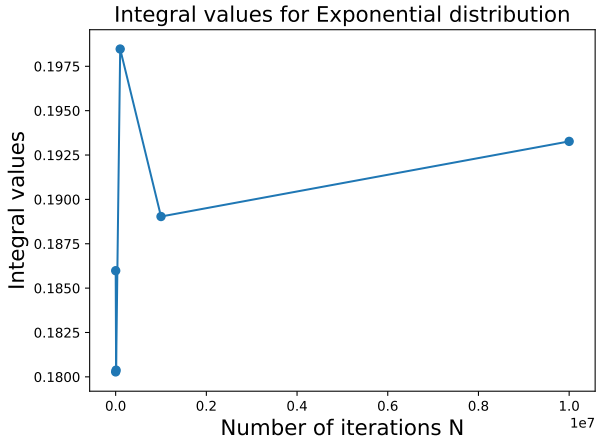
N	Calculated integral	Variance	Time spent [s]
10^4	0.111911	0.000472	0.001852
10^5	0.210185	0.009714	0.01534
10^6	0.190882	0.023387	0.1736
10^7	0.193220	0.017242	2.122

Table 4: Results for Monte Carlo integration of the integral in spherical coordinates.

N	Calculated integral	Time spent [s]
10^4	0.184573	0.00254
10^5	0.191822	0.02253
10^6	0.191636	0.2445
10^7	0.193091	2.533

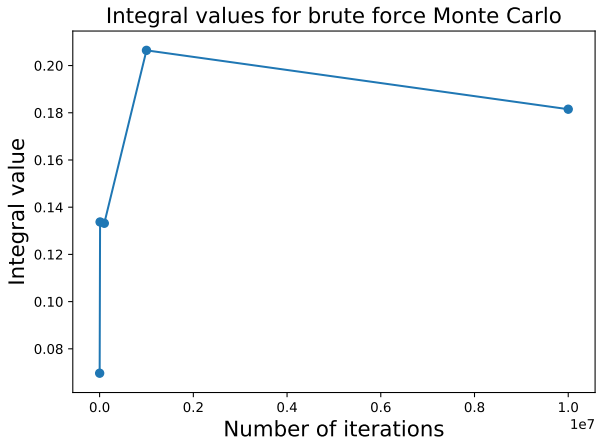
Table 5: Results for parallelization with 2 processors.

N	Calculated integral	Variance	Time spent [s]
10^4	0.189775	0.023090	0.0186
10^5	0.202494	0.053686	0.0624
10^6	0.191768	0.042251	0.1395
10^7	0.193470	0.044060	1.2428



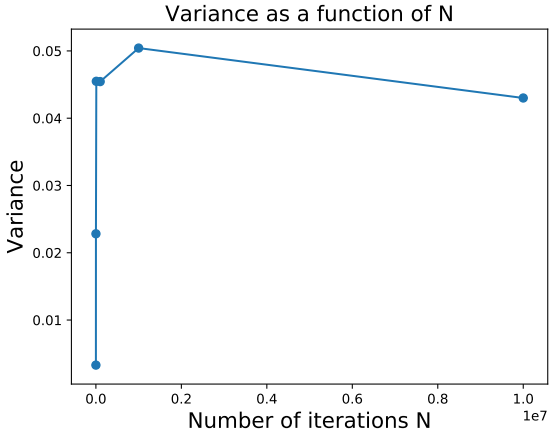
Integration/exp$_i nt.pdf$

Figure 2: Time to run the algorithm shown as a function of iterations N for the Monte Carlo Integral using the exponential function



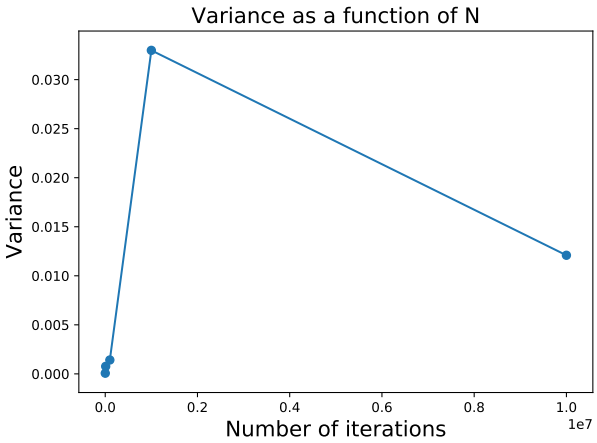
Integration/brute$_i nt.pdf$

Figure 3: Value of the integral computed as a function of N for the Monte Carlo method using the exponential distribution.



Integration/variance.pdf

Figure 4: Variance of the Monte Carlo integral using the exponential distribution.

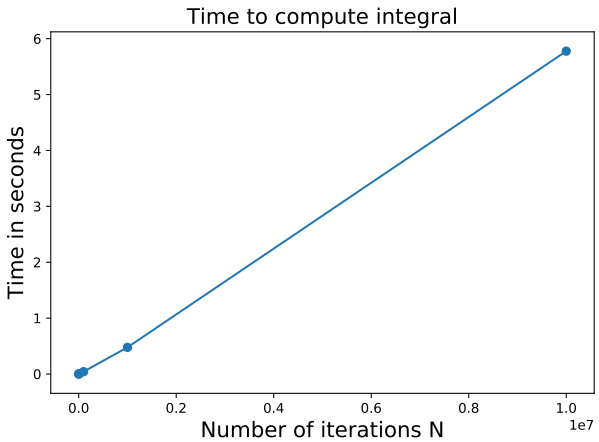


Integration/brute$_v$*ar.pdf*

Figure 5: Variance as a function of the number of iterations when we used the normal distribution for all variables.

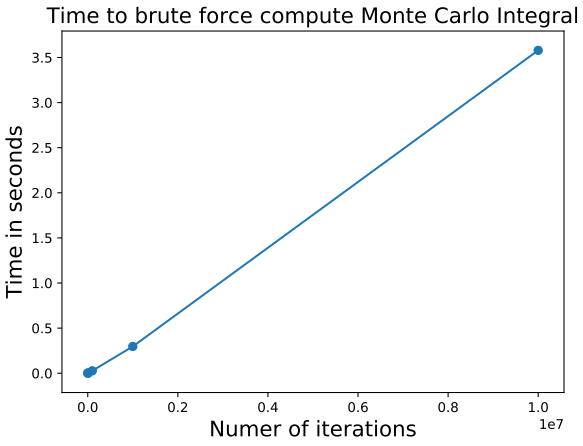
Integration/parallel$_t$*ime.pdf* `Projects/Numerical Integration/parallel_time`

Figure 6: The time it took to compute the integral values using Monte Carlo integration with the exponential distribution for the radial components of the integral when we parallelized the code. 4 cores were used when we parallelized



Integration/exp$_t$*ime.pdf*

Figure 7: Time it took to compute the integral values using Monte Carlo integration with the exponential distribution for the radial components of the integral when we did not parallelize the code.



Integration/brute$_t ime.pdf$

Figure 8: Time to compute the integral using the brute force approach.

4 Discussion

4.1 Gaussian Quadrature

The first thing that was noticed for this method and specially since the integral we wanted to compute was six dimensional. Was the time spent to get a result for a high N. This is clear when for $N = 25$ we had to spent over an hour of runtime to get our result. Another thing that is worth noticing is that for our case the Legendre integration converges much more slowely than the laguerre. This might be because we approximate infinity in the cartesian coordinates with a finite limit of λ. This gives us a clear indication that for lower Ns it is much better to approximate the integral with Laguerre polynomial for the radial part. This also makes sense considering that laguerre polynomials are defined for $[0, \infty)$ while the Legendre polynomials are determined in the interval $[-1, 1]$

4.2 Parallelization

Both the program "SmarterMonteCarlo.py" , and "MonteCarloDone.py" are small programs and does not use much time to run. None of them include for loops and are both vectorized. Parallelization is useful if a program has multiple function calls and the calls are independent of eachother. The process of scattering and gathering data would in this case not be useful because of the low runtime of the programs, it might even make the programs slower. An argument can be made for not parallizing a program if it is contemptuously fast because the time it takes to distribute computations

and gather them again is sometime larger than the time saved by parallizing. The amount of available processors is, as expected, crutial for the times saved by parallizing. For a pc with two cores the the speedup of "Parallel-MonteCarlo.py" was 2.2, and for a computer with four cores the speed up was 4.

4.3 Monte Carlo Integration

We see that when we compute the Monte Carlo integral using the exponential function we converge on the correct value of the integral faster than we did for the uniform distribution. We see that it approaches the correct value of the integral slightly faster than the uniform distribution. This is to be expected as the behavior of the function we want to integrate in the r_1 and r_2 dimensions behave more like exponential functions than they behave like straight lines.
We are not able to say too much about the differences in variance between the brute force approach and the exponential because the variance varies too much between each time we run the program. In the plots we've included we can see that we generally get a higher variance when we compute the integral using the exponential approach, but this seems like an unreasonable result and is likely a fluke. The exponential approach should have less variance as the exponential function should do a better job of representing the values of the function we want to integrate. We also see that when neither is parallelized the brute force approach is slower than the exponential approach. This is probably because, by using the exponential distribution, we have removed several computations from our program.

We see that for parallelization results in a significant speedup if N is large enough. The computer which ran the simulations which resulted in the plots of the time it took to run the integration program had 4 cores and the program used all of them. In theory this could result in a speedup of 4. Looking at the plots, we see that the run time of the program increases linearly as a function of N both for the parallelized and the non-parallelized code. We see that for lower values of N, the parallelization does not decrease the run time of the program at all, but actually increases it slightly. However, for larger N the parallel program runs significantly faster than the normal program. This is to be expected, as it takes some time to divide up the task to the different processors, which is not then gained back because there are so few computations to be performed. For larger N the parallelization benefit us because the time it takes to divide up the task of running the program is small compared to the time we save by having multiple cores working in parallel. For the largest value of N we actually gain a speed for the parallel program which is more than 4 times greater than the time for the non-parallel program, which is greater than the theoretical

speedup of 4 and therefore is clearly a fluke. However, in spite of some uncertainty in our results, it does demonstrate the benefit of running heavier computations in parallel.

5 Conclusion

In conclusion we then See that for the Gaussian Quadrature methods the Laguerre polynomials are a better approximation for our integral since we need less integration points and therefor less time to get a good estimate.

We see that using the normal distribution to compute the Monte Carlo is both slower to converge for an equal number of iterations N and takes a longer time to compute the integral for the same number N using the exponential distribution for some of the variables. Parallelization is useful for speeding up the program if we have enough computations which needs to be done.