Numerical Integration - The Monte Carlo Method

Christian B. Molina

March 19, 2021

Abstract

Monte Carlo integration is a method of numerical integration in which a uniform distribution of random numbers between bounds are sampled into the integrand. As more values are sampled, the approximated integral gets closer to the exact value of the actual integral. This method differs greatly from other methods of numerical integration like the rectangular rule and trapezoidal rule that use a deterministic approach - rather Monte Carlo integration employs a non-deterministic approach to solving the integral. Instead of a direct input defining a predictable output, Monte Carlo integration relies on the random probability distribution to greatly define it.

In this report, an algorithm to apply Monte Carlo integration will be outlined and then applied to three definite integrals - a one-dimensional Gaussian function, a three-dimensional function describing a screened charge distribution, and an improper integral.

1 Introduction

1.1 Monte Carlo Integration

The effectiveness of Monte Carlo integration stems from the usage of certain sampling techniques to choose points to evaluate the integrand *f(x)*, rather than evaluating at predetermined evenly spaced points implemented in rectangular, trapezoidal, and Simpson's rule, or at points that are determined by the roots of certain order Legendre polynomials in Gauss-Legendre integration. This simplifies the process of integrating improper and multidimensional integrals.

In this report, random sampling of a bounded uniform distribution will be utilized. The interval will be defined from [0,1], since most random number generators are designed for such a range. For integrals that have lower limits a or upper limits b that are different from 0 or 1, the following transformation can be applied:

$$x \longrightarrow \frac{1}{b-a}(x-a) \tag{1}$$

Taking a sampling size with a large N, the average distance between two neighboring points can be defined as:

$$h = \frac{1}{1-N} \underset{largeN}{\longrightarrow} \frac{1}{N} \tag{2}$$

Therefore, the value of the integral can be defined as:

$$f_{[a,b]} = \frac{1}{b-a}\int_a^b f(x)dx \underset{N\to\infty}{\longrightarrow} \frac{1}{N}\sum_{i=1}^{N} f(x_i) \tag{3}$$

If sampling is truly random, the number of points selected in a given region is independent of the value x [1]. The interval [0,1] can then be divided into an arbitrary number of equal-sized subintervals - with each subinterval the number of points have the same statistical fluctuations if large enough samples are taken.

To determine if our sample size is large enough, an iterative strategy can be employed. If a sample size of N_1 is taken to evaluate the integral two different times, the results can be compared by calculating the difference between the results. If a tolerance ϵ is defined and the difference between the results do not satisfy the tolerance, then a greater sample size can be taken such as $N_2 = 2 * N_1$. Repeating this process again with greater sample sizes will eventually meet the condition of the tolerance.

This iterative strategy can be applied to one-dimensional, multi-dimensional and improper integrals, but must be updated to their individual conditions.

1.2 One-Dimensional Integrals - Normal Probability Distribution

For applying Monte Carlo integration for a one-dimensional integral, the normal probability function will be used as an example:

$$A(x) = \frac{1}{\sqrt{2\pi}} \int_{-x}^{x} e^{\frac{-t^2}{2}} dt \tag{4}$$

Since the sampling for the Monte Carlo integration is bounded to $[0, 1]$, the lower and upper limit of the normal probability function can be adapted to correspond with the bounds of $[0, 1]$ by simply applying symmetry and having the upper bound be defined as $x = 1$, therefore the integral is now:

$$A(x = 1) = \sqrt{\frac{2}{\pi}} \int_{0}^{1} e^{\frac{-t^2}{2}} dt \tag{5}$$

A is the area underneath the normal distribution curve, therefore the expected value is:

$$A(x = 1) = 2(\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{1} e^{\frac{-t^2}{2}} dt - \frac{1}{2}) = 0.6826895 \tag{6}$$

1.3 Multi-dimensional Integrals - Yukawa Charge Distribution

Since multi-dimensional integrals require integration across all of N-space, the same method applied to the one-dimensional integral can be applied to N-dimensional integrals. In this case, for a three-dimensional integral, the average evaluated throughout all of space is defined as:

$$I = \int_{e}^{f} \int_{c}^{d} \int_{a}^{b} f(x)f(y)f(z)dxdydz \tag{7}$$

$$I = \sum_{i=1}^{N} f(x_i)f(y_i)f(z_i)\Delta v \tag{8}$$

in which

$$\Delta v = \frac{(b-a)(d-c)(f-e)}{N} \tag{9}$$

The benefit of the Monte Carlo method is that there is no need to integrate over all space, a random sampling with a sufficiently large N of n the number of integration variables[1].

Applied to the Yukawa charge potential confined to a cubic volume of length 2 on each side where the charge distribution $\rho(x, y, z)is$:

$$\rho(x, y, z) = \frac{e^{-r}}{8\pi} \tag{10}$$

where

$$r = \sqrt{x^2 + y^2 + z^2} \tag{11}$$

The total charge inside this volume is then:

$$Q = \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} \frac{e^{-r}}{8\pi} dxdydz \tag{12}$$

symmetry similar to equation (1) can be applied to all variables, therefore the limits of integration for all space are $[0, 1]$ is defined as:

$$Q = \int_{0}^{1}\int_{0}^{1}\int_{0}^{1} \frac{e^{-r}}{\pi} dxdydz \tag{13}$$

$Q \approx 0.4/\pi$ is the expected result, and an accuracy of .0002 should be reached by $N = 8000$ sample points. In this report, the solution to the Yukawa charge distributions confined to a box will begin with an initial random sampling size of $N = 10$, tolerance $\epsilon = 10^{-4}$.

1.4 Improper Integrals - Arcsin

Improper integrals have points or limits that produce a singularity. In the case of the trigonometric identity $\arcsin x$, the singularity is defined at when $x = 1$. The Monte Carlo integration can be adapted for this case.

Since Monte Carlo integration selects random numbers throughout $[0, 1]$ or even $[a, b]$ in which a or b could be points of singularity - in the case they are selected, those values can be discarded and have a new random value selected. The benefit of Monte Carlo integration is that significant approximation of the integral is not loss when the individual points is discarded. Instead the accuracy of Monte Carlo integration relies on the amount of points sampled.

$$I(B) = \int_{0}^{B} \frac{dx}{\sqrt{1 - x^2}} = arcsin(x) \tag{14}$$

2 Methods

2.1 Algorithm: One-Dimensional Monte Carlo Integration

The following algorithm will utilize the logic of equation (3). Due to the iterative behavior described in Section 1.1, this algorithm would have to be in a constant loop until the condition tolerance condition ϵ is met. It is also ideal to be able to store previous values in which the difference can be calculated.

Algorithm 1: Monte Carlo Integration for One-Dimensional Integral

```
Result: integralAvg
integralResults: list of zeros with size 3;
seed: to initialize random.random();
integralAvg = 0;
sum = 0;
counter = 0;
N = 40;
ϵ = pow(10,-5);
while True do
    counter += 1;
    while i = 1 ≤ 3, i++ do
        while n = 0 < N do
            x = random.random();
            sum += normProb(x);
        end
    end
    integralResults[i] = (1/N) * sum ;
    diff = |integralResults[1]− integralResults[2]|;
    if diff ≤ ϵ then
        break;
    else
        N = 2 * N;
    end
end
```

A seed for the random number generator must be defined and initiated. In this example, the tolerance ϵ will be defined as 10^{-5} and the beginning number of random points will be defined as 40. The major while will run until the difference of the integralResults[1] and integralResults[2] is less than or equal to the tolerance.

It is worth mentioning that *normProb* is a direct translation of the integrand and coefficients of equation (5).

2.2 Algorithm: Multi-Dimensional Monte Carlo Integration

Algorithm 2: Monte Carlo Integration for Three-Dimensional Integral Yukawa

```
Result: integralAvg
integralResults: list of zeros with size 3;
seed: to initialize random.random();
integralAvg = 0;
sum = 0;
counter = 0;
N = 40;
ϵ = pow(10,-5);
while True do
    counter += 1;
    while i = 1 ≤ 3, i++ do
        while n = 0 < N do
            x = random.random();
            y = random.random();
            z = random.random();
            sum += yukawa(x,y,z);
        end
    end
    integralResults[i] = (1/N) * sum ;
    diff = |integralResults[1]− integralResults[2]|;
    if diff ≤ ϵ then
        break;
    else
        N = 2 * N;
    end
end
```

The algorithm above has the same structure as Algorithm 1. The only difference is how many random points are implemented during the evaluation of the integrand. Since the Yukawa charge distribution is defined in three dimensions, three independent variables for random sampling must be defined.

The function yukawa is also a direct translation of equation (8).

2.3 Algorithm: Improper Integration

Algorithm 3: Monte Carlo Integration for Improper Integral

```
Result: integralAvg
integralResults: list of zeros with size 3;
seed: to initialize random.random();
b: upper limit, must be less than 1.;
integralAvg = 0;
sum = 0;
counter = 0;
N = 40;
ϵ = pow(10,-5);
while True do
    counter += 1;
    while i = 1 ≤ 3, i++ do
        while n = 0 < N do
            x = random.random();
            while x == 1 or x ≥ b do
                x = random.random()
            end
            sum += trigArcSin(x);
        end
    end
    integralResults[i] = (1/N) * sum ;
    diff = |integralResults[1]− integralResults[2]|;
    if diff ≤ ϵ then
        break;
    else
        N = 2 * N;
    end
end
```

Lastly, this algorithm still follows a similar structure, but with a case in which the random value generated for x is checked for the case in which it is a singularity. If the random number is equal to 1, or greater than or equal to the upper limit b, then new sample value will be regenerated. The upper limit must also be less than b, since the random number generator only generates values between [0,1].

3 Discussion

The following figures and graphs display the trend of Algorithms 1, 2 and 3. As iteration increases and the number of points increase, the approximated value gets closer to the exact value. In each table, the percent difference is displayed.

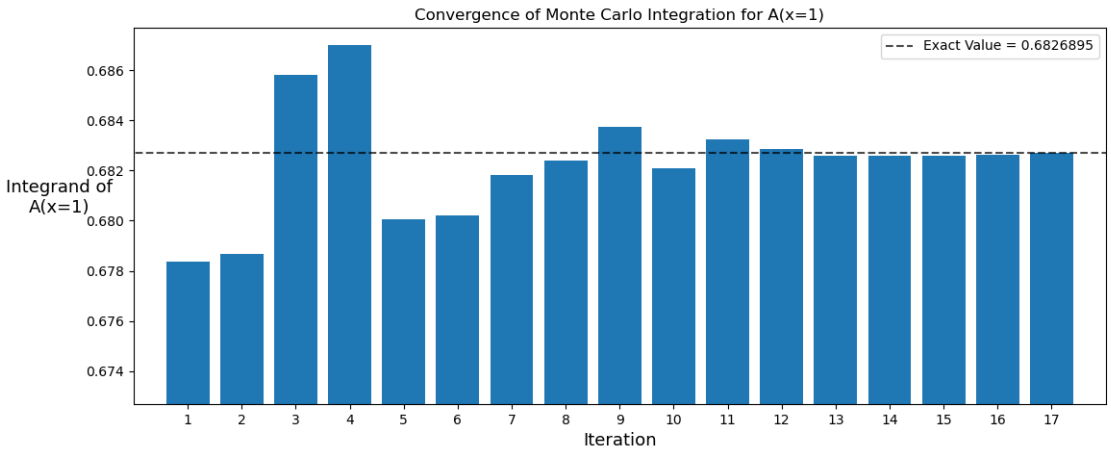

Figure 1: The convergence of Algorithm 1 approximating the integral of *A(x=1)*

Table 1: Approximation of a Monte Carlo Integration for $A(x) = \sqrt{\frac{2}{\pi}} \int_0^1 e^{\frac{-t^2}{2}} dt$

Iteration	# of Points	$A(x = 1)$
1	40	0.6783558982609414
2	80	0.6786819472624124
3	160	0.6858199499344058
4	320	0.6869904159426077
5	640	0.6800648820982704
6	1280	0.6802184608692039
7	2560	0.6818132538095749
8	5120	0.6824076723724196
9	10240	0.6837210785252702
10	20480	0.682075408910586
11	40960	0.683225180957008
12	81920	0.6828445021426981
13	163840	0.6825845978458642
14	327680	0.6826057343215595
15	655360	0.6826062991988391
16	1310720	0.6826395563173886
17	2621440	0.6827022565933677
Exact Value		0.6826895
Percent Error		0.0018685791077417309 %

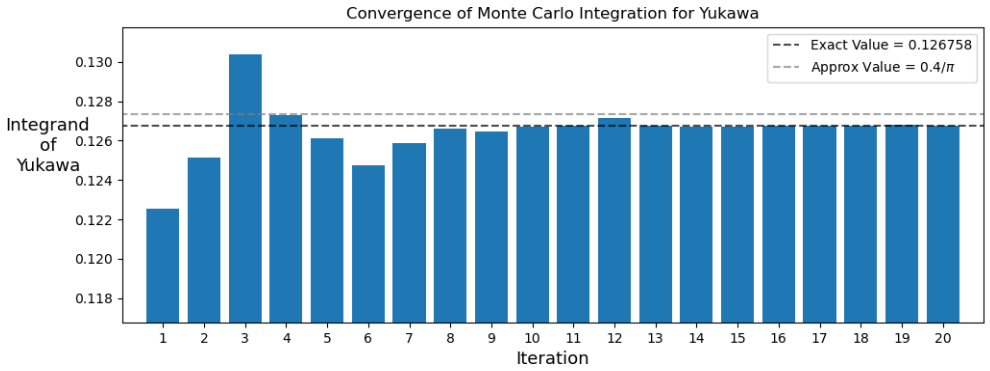

Figure 2: The convergence of Algorithm 1 approximating the integral of *Q(1,1,1)*

Table 2: Approximation of a Monte Carlo Integration for $Q(x,y,z) = \int_0^1 \int_0^1 \int_0^1 \frac{e^{-r}}{\pi} dxdydz$

Iteration	# of Points	$Q(x,y,z)$
1	10	0.12252133730716008
2	20	0.12513573730427444
3	40	0.13039211718036856
4	80	0.12731317249598925
5	160	0.12613475561212287
6	320	0.12474285635241708
7	640	0.12586994385829303
8	1280	0.12662736862127963
9	2560	0.12643789748905324
10	5120	0.12669226895489638
11	10240	0.12673551077290815
12	20480	0.12713683075176996
13	40960	0.12674986188900222
14	81920	0.1267008221194169
15	163840	0.1267137167007863
16	327680	0.1267477656759774
17	655360	0.12673103437225258
18	1310720	0.12675348680372833
19	2621440	0.12677982896690812
20	5242880	0.12676003476583658
Exact Value		0.126758
Percent Error		0.0016052366214145805 %

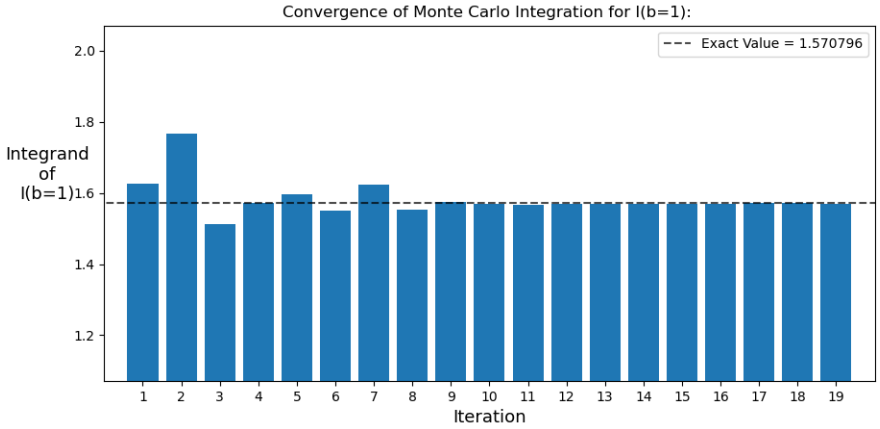

Figure 3: The convergence of Algorithm 3 approximating the integral of *I(b=1)*

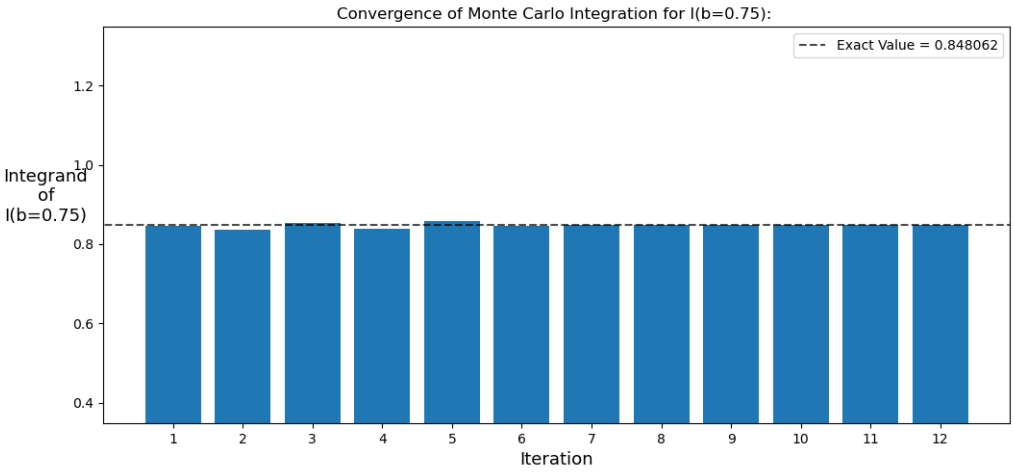

Figure 4: The convergence of Algorithm 3 approximating the integral of *I(b=0.75)*

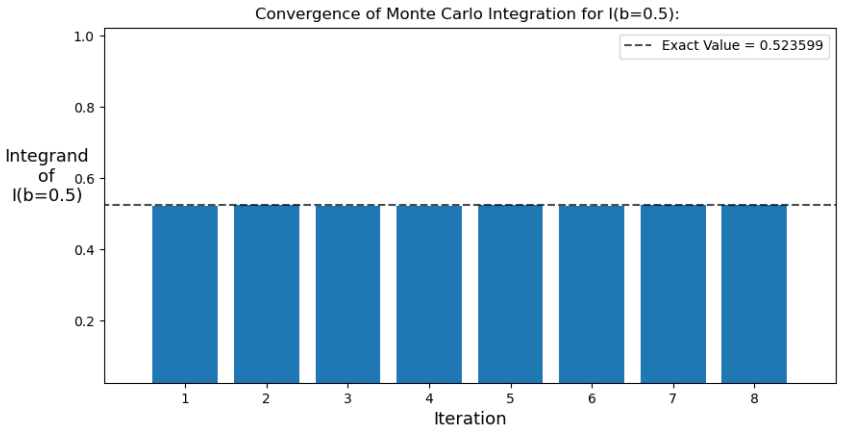

Figure 5: The convergence of Algorithm 3 approximating the integral of *I(b=0.5)*

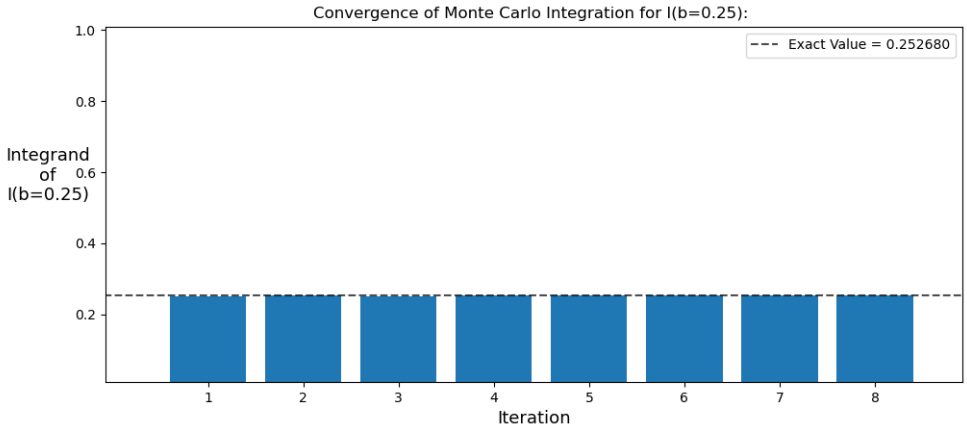

Figure 6: The convergence of Algorithm 3 approximating the integral of *I(b=0.25)*

Table 3: Convergence of a Monte Carlo Integration for $I(1) = \int_0^1 \frac{dx}{\sqrt{1-x^2}}$

Iteration	# of Points	$I(x=1)$
1	40	1.6258447254954138
2	80	1.766804627935255
3	160	1.5132403074431098
4	640	1.5730378188429648
5	1280	1.595066192044791
6	2560	1.5495706932824924
7	5120	1.6230247109421394
8	10240	1.552518596879179
9	20480	1.5747638862847197
10	40960	1.5685092804476124
11	81920	1.5674919232508437
12	163840	1.5698619417225843
13	327680	1.569666641002282
14	655360	1.568954860388696
15	1310720	1.5689527994372923
16	2621440	1.5694338693755476
17	5242880	1.5706485250002555
18	10485760	1.5709506441753998
Exact Value		1.5707963267948966
Percent Error		0.023716804022156576 %

Table 4: Convergence of a Monte Carlo Integration for $I(0.75) = \int_0^{0.75} \frac{dx}{\sqrt{1-x^2}}$

Iteration	# of Points	$I(x = 0.75)$
1	40	0.846777442345596
2	80	0.8354334055815846
3	160	0.8528094949058389
4	320	0.8384644981211924
5	640	0.8569885948727841
6	1280	0.8451241228234809
7	2560	0.8490269724263056
8	5120	0.8477814649121527
9	10240	0.8485568206138918
10	20480	0.848023212772286
11	40960	0.8476871636541179
12	81920	0.84826378410736
Exact Value		0.848062078981481
Percent Error		0.02378424066800309 %

Table 5: Convergence of a Monte Carlo Integration for $I(0.50) = \int_0^{0.50} \frac{dx}{\sqrt{1-x^2}}$

Iteration	# of Points	$I(x = 0.50)$
1	40	0.5216678878118393
2	80	0.5245461276852734
3	160	0.5230433085124537
4	320	0.5231140638792202
5	640	0.52440509222802
6	1280	0.5226907711323325
7	2560	0.5235321093107281
8	5120	0.5236906827240863
Exact Value		0.5235987755982989
Percent Error		0.01755296804930607 %

Table 6: Convergence of a Monte Carlo Integration for $I(0.25) = \int_0^{0.25} \frac{dx}{\sqrt{1-x^2}}$

Iteration	# of Points	$I(x = 0.25)$
1	40	0.25209005366508364
2	80	0.25274711387596727
3	160	0.25233742818143223
4	320	0.2527559117738851
5	640	0.2526846607345739
6	1280	0.25266134309215704
7	2560	0.25272855471519884
8	5120	0.25265528776299867
Exact Value		0.25268025514207865
Percent Error		0.009881017045015119 %

4 Conclusion

Monte Carlo integration proves to be efficient in terms of calculating multi-dimensional and improper integrals. The only issue that arises is the computation time for the algorithms when sampling higher values of N. This can be mitigated by setting a counter to limit iterations.

References

[1] S.S.M Wong.*Computational Methods in Physics and Engineering*