Numerical Computation Project:

Monte Carlo Methods

Bella Longo, Harika Kondur, Harley Ewert, and Zach Cook

Department of Computer Science, University of Colorado Boulder

CSCI 3656: Numerical Computations

Dr. Divya E. Vernerey

December 13th, 2023

1 Abstract

This is an introduction to Monte Carlo Methods, a type of numerical method meant to estimate a random value by repeated random sampling. In this paper, we will first cover the basics of what Monte Carlo Methods are, briefly explain the history of how they came to be used today, and discuss their accuracy. We will then cover how these methods are used in the real world in fields such as Finance and Computer Science. We will then cover how they relate to other numerical methods used for both integration and root finding. Finally, we will cover how Monte Carlo Methods can be coded.

2 Introduction

In the world of Numerical Methods, Monte Carlo Simulations are unusual in that it is difficult to pin a specific definition down without leaving out certain examples [3]. While other methods are so well defined that one must follow a specific algorithm, the open-ended nature of Monte Carlo Methods defy formalization. With that being said, a robust broad definition would be "Monte Carlo simulation is the term applied to stochastic simulations, either discrete, real-time, or some combination thereof, that incorporate random variability into the model." [2]

Despite the differences between different applications of the Monte Carlo Method, the crucial element they all share is their model. The model in a Monte Carlo simulation is a representation of the system that is being studied. For example, if one was trying to predict how different quantities of a certain company stock would perform, they would need to construct

a model that would act as a faux-stock market through its inputs, outputs, and internal calculations. According to Robert L. Harrison [3], "Key points to consider in defining a model are:

- what are our desired outputs?
- what will these outputs be used for?
- how accurate/precise must the outputs be?
- how exactly can/must we model?
- how exactly can/must we define the inputs?
- how do we model the underlying processes?"

By answering these questions, a numeric modeler will receive everything they need to know about what constraints their model will have, which should give them a good idea where to start their model, so that it is as accurate and useful as possible.

An important aspect that the list by Harrison reveals is the variable nature of the accuracy of Monte Carlo Methods. Because models change drastically from one application to another, the accuracy really depends on the model, and there is no uniform way to measure the error. W.F Bauer puts it best in his paper, The Monte Carlo Method: "As in almost all numerical processes, only an approximation to the correct answer is obtained. In this case, instead of the primary source of error being due to numerical round-off, the primary source of error is due to the fact that only a finite sample can be taken. It follows, of course, that the degree of accuracy depends on the sample size." [1] Due to modern computing capabilities, obtaining a large sample size is relatively easy, and computationally cheap, making it a strong numerical

tool that will only become stronger as technology advances. For this reason understanding Monte Carlo Methods and their applications is becoming increasingly important.

3 Background

The first instance of Monte Carlo simulation was performed in the 18th Century, by French scientist Georges Louis LeClerc, Comte de Buffon. While it is known today as "Buffon's needle experiment", Buffon purportedly tossed baguettes over his shoulder onto a lined, tiled floor to estimate π. He had previously proved that a randomly placed segment the same length as the spaces between a lined background would intersect with probability $2/\pi$. So after n tosses, Buffon estimated π as (2 * n) / X, where X is the number of baguettes that intersected a line made from the tile. [3]

After Buffon's experiment there were many other instances of scientists using stochastic simulation in the 19th and early 20th century. However, like Buffon's needle, most of these instances were for deterministic problems where the answer was already known. Monte Carlo simulations as we use them today can be traced to John von Nuemann and Stanislaw Ulam in the early to mid 1940's. The pair was working on the Manhattan Project during World War 2, and wanted to study how neutrons traveled through radiation shielding, and found that Monte Carlo Methods were a perfect tool for this. Even after the neutron study, Nuemann and Ulam found the method so useful, they used it in a variety of other nuclear weapon problems. Most important of all, the duo is responsible for the name "Monte Carlo" methods, after they drew a parallel from the random nature of the method to the famous Monte Carlo Casino in Monaco.[3]

After its use in World War 2, Monte Carlo Methods grew more popular due to how well it works for situations where real world experimentation is impractical or impossible. While it may have its disadvantages, the fact the simulations not only allow researchers to study complex systems for relatively cheap, but also repeat experiments in ways not possible in the real world make this a tool used all over the different fields of sciences.

4 Real-World Applications

Monte Carlo methods are useful in simulating performance and solving problems that involve sampling, optimization, and estimation. Some examples of how Monte Carlo methods are used in real life include:

4.1 Supply Chain and Logistics

Monte Carlo simulations are used in the field of industrial engineering, specifically supply chain and logistics management. Key applications include optimizing inventory levels, conducting demand forecasting analysis, and assessing supply chain risks. By simulating these scenarios, industrial engineers can better evaluate the impact of uncertainties, optimize logistical operations and improve the overall supply chain efficiency.

4.2 Economics and Finance

Monte Carlo methods are widely used to estimate the value of stocks, investments, and other financial instruments. By simulating different scenarios, investors can use Monte Carlo methods to better understand the risks associated with a particular investment across limitless

model assumptions.

4.3 Physical Sciences

Monte Carlo techniques play a pivotal role in materials science, contributing to the development and analysis of new materials and structures, such as organic LEDs and solar cells. In particular, Monte Carlo techniques play a key role in virtual materials design, where experimental data is used to produce stochastic models of materials, which can then be used to perform simulations and numerical experiments. This virtual approach generates additional data, allowing a deeper exploration of material behavior under various production parameters. [4]

4.4 Computer Science

Monte Carlo methods also have many uses in computer science, such as optimization, artificial intelligence, and machine learning. For example, they can be used to train a neural network, or to optimize the parameters of a machine learning model.

Overall, Monte Carlo methods are used to solve problems that involve uncertainty and multiple outcomes. By running multiple simulations, these methods can help identify the outcome most likely to occur and help people make informed and data-driven decisions.

5 Root-Finding

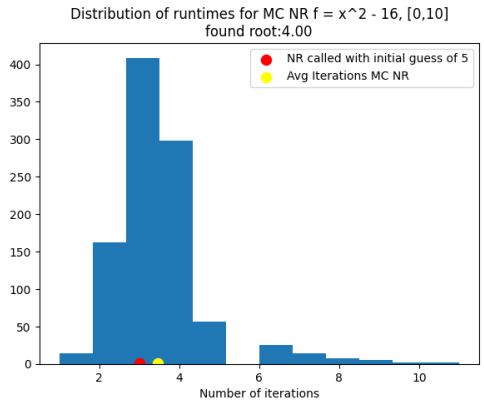
Monte Carlo relates to root finding in that it is used to approximate a first initial guess. If you have an interval $[a, b]$ in which you are trying to find a root, then you can generate a

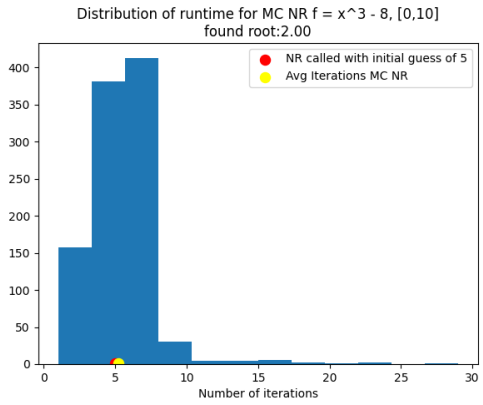
random number c which will be used as an initial guess. "If the random numbers be near the root [] this algorithm will converge rapidly", and if the number is a poor estimate it will converge slowly [5]. We can draw some interesting conclusions regarding the long term performance of the Monte Carlo method given that it depends on random chance. Over a large number of simulations, the number of best case estimates will be equal to the number of worst case estimates. This means that using Monte Carlo for root finding, if the application would need to be run over and over, would lead to average performance, and is actually quite useful. Average performance is nothing to be scoffed at in the world of computation.

For example, Newton Raphson root finding is defined as

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Because the whole algorithm depends on the first value plugged into x_i, an initial guess is quite important for determining the convergence of the algorithm. If the initial guess is far off it could take significantly longer to converge. A basic example of how Monte Carlo root finding could be implemented is found in python script in the appendix, but on a high level the code samples from a uniform distribution that covers the range of our bounds $[a, b]$, then uses this random sample as an initial guess for the Newton Raphson root finding method. The resulting histograms show the distribution of the number of iterations the Newton Raphson root finding method takes to find the root, given 1000 different initial random guesses, and an error tolerance of 0.001.





From these graphs we can see that the MC Newton Raphson root finding performs on average only slightly worse than an initial guess that is actually quite good. In these simulations the yellow dot represents the runtime of the Newton Raphson algorithm if the initial guess is simply the midpoint of the interval. In this case this works fine because the root is close to the midpoint, but in the real world the root may be closer to one of the edges. In this case a fixed midpoint for every trial could drag down performance, whereas using MC would lead to average performance in most scenarios.

From this demonstration we can see that using Monte Carlo simulation in root finding may be useful in certain situations. If finding a good initial guess would take a lot of time, it might be useful to randomly choose an initial guess. However if you can reliably and cheaply choose a good initial starting point, that would outperform Monte Carlo.

5.1 Root-Finding Code

```
#----------------MONTE CARLO NEWTON RAPHSON ROOT FINDING---------------------

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd


def y1(x):

    #function one: y = x^2 -16

    return x**2 - 16

def y1p(x):

    #y' = 2x

    return 2*x

def y2(x):

    #function two: y = x^3 - 8

    return x**3 - 8

def y2p(x):

    #y' = 3x^2 - 8

    return 3*x**2


def newton(f, fp, pn, err, n=0, lim = 1000):

    # ----------------NEWTON RAPHSON ROOT FINDING--------------

    # f - function in question
```

```
    # fp - derivative of that function
    # pn - the initial approximation
    # err - your error tolerance |pn - pnp1| < err
    # lim - uperbound for number of recursive calls
    if abs((f(pn)/fp(pn))) < err or n == lim:
        return [pn,n]
    return newton(f, fp, pn - (f(pn)/fp(pn)), err, n + 1, lim)


#Interval [a,b]
a = 0
b = 10


num_iterations_y1 = []
num_iterations_y2 = []
values_y1 = []
values_y2 = []


#Following Code is reponsible for MC Root Finding Trials
for i in range(1000):
    value_y1, iters_y1 = newton(y1, y1p, float(np.random.uniform(a,b,1)), 0.001)
    value_y2, iters_y2 = newton(y2, y2p, float(np.random.uniform(a,b,1)), 0.001)
    values_y1.append(value_y1)
    values_y2.append(value_y2)
```

```
num_iterations_y1.append(iters_y1)

num_iterations_y2.append(iters_y2)
```

6 Integration

In the domain of Monte Carlo applications, using it as a tool to integrate is at the forefront. When faced with challenging integrals with high dimensionality or even irregular functions, Monte Carlo is a sure and accurate method to use. Deterministic methods, such as Trapezoidal Rule:

$$\int_{x_2}^{x_1} f(x)\,dx \quad \frac{h}{2}\left[y_o + y_2\right]$$

or Simpson's Rule:

$$\int_{x_2}^{x_1} f(x)\,dx \quad \frac{h}{3}\left[y_o + 4y_1 + y_2\right]$$

use the shape of the function to make an accurate estimate for the integral, especially for smooth curves. These two methods utilize the information at various points in the function to make their estimation, but what if the function is not that simple? In scenarios like these, the two methods face the 'curse of dimensionality', when computational cost increases exponentially with a more complicated function. This is when Monte Carlo takes the reins taking a non-deterministic approach. Unlike the deterministic approach, the non-deterministic approach utilizes the aspect of randomness. With each iteration, there is a new view of the function, allowing it to adapt to the function complexity and not to be focused on a specific function behavior that can come with function oscillations.

To use Monte Carlo for numerical integration, a function $f(x)$ must first be defined that will be integrated across the region $[a, b]$. A random set of x-values denoted by x_i must be

generated and must be within the integration bounds $[a, b]$ of sample size N. Now for each random sample x_i, evaluate the function $f(x_i)$. This result must be scaled to serve as the width of the integral, achieved by multiplying $f(x_i)$ by $(b-a)$. To get our integral estimate, we must average the results of each x_i, with the final result being:

$$\frac{1}{N}\sum_{i=1}^{N} f\left(x_i\right) \cdot (b-a)$$

To better illustrate this method, high-computing languages like Julia or Python can be utilized for efficient computation. The following Julia code demonstrates the methods described above to estimate the integral of a function $f(x)$ using Monte Carlo.

```
function monte_carlo_integration(f, a, b, n)
    x = rand(Float64, n) * (b - a) + a
    y = f(x)
    integral_approx = (b - a) * sum(y) / n
    return integral_approx
end
```

In this Julia function, 'f' represents the function that will be integrated, with 'a' and 'b' being the integration bounds, and 'n' being the sample size.

To demonstrate the efficiency of the non-deterministic approach the algorithm takes, we will integrate the function $f(x) = e^{-x^2}$ over the interval $[0, 1]$ with a sample size of 1000 and compare the result to the result that Composite Trapezoidal given by:

$$\int_a^b f\left(x\right) dx \;=\; \frac{h}{2}\left[f\left(a\right) + 2\sum_{j=1}^{n-1} f\left(x_j\right) \;+ f\left(b\right)\right]$$

When using a step size of 100 for the Composite Trapezoidal Rule, it gives

$$\frac{100}{2}\left[f\left(0\right)+2\sum_{j=1}^{99}f\left(x_j\right)\ +f\left(2\cdot\pi\right)\right]$$

and a result of 0.8959, while Monte Carlo got a result of 0.8865. The actual result of the integral was 0.8862, proving Monte Carlo not only being more computationally efficient, but also more accurate than the Composite Trapezoidal Rule. Monte Carlo was able to adapt to the oscillations, as well as high dimensionality to get the accurate approximation.

Error is an important factor that must also be considered when performing Monte Carlo for integration. The result of the algorithm is subject to fluctuations due to the randomness of the non-deterministic approach. As the sample size increases, the result converges to the true value of the integral, decreasing error. However, errors will never disappear entirely no matter the sample size due to the aspect of randomness. Error in the algorithm is expressed by:

$$MCE\left(\phi_R\right)=\sqrt{Var\left(\phi_R\right)}$$

with ϕ_R representing the estimate and of ϕ with R replications. Due to sample size being correlated to error, computational efficiency must also be taken into account.

It is evident that the choice between a more deterministic approach and a non-deterministic one depends on the complexity of the function. Deterministic approaches like the Trapezoidal Rule and Simpson's Rule are sufficient for smooth curves, but falter when functions become increasingly complex, letting Monte Carlo shine. Its ability to adapt to high-order and oscillating functions allows it to navigate the function with ease, getting a fresh perspective with each iteration, and getting a more accurate, as well as well-rounded, result.

7 Markov Chain Monte Carlo (MCMC)

The Monte Carlo method, while powerful and versatile, has some shortcomings in the field of probability modeling and Bayesian statistics that can be addressed by Markov Chain Monte Carlo (MCMC) methods. Firstly, traditional Monte Carlo sampling may not work well for high dimensions probabilistic models, due to the curse of dimensionality, where the volume of the sample space increases exponentially with the number of parameters. It also assumes that each random sample drawn from the probability distribution is independent and can be independently drawn which is typically not the case nor efficient for Bayesian and graphical probabilistic models.

The solution to sampling probability distributions in high dimensions is to use Markov Chain Monte Carlo, or MCMC for short. The idea behind MCMC is to impose a dependency between samples, i.e. samples are drawn from a probability distribution by constructing a Markov Chain, where the next sample that is drawn from the probability distribution is dependent upon the last sample that was drawn.

Markov Chain Monte Carlo is a method used in probability statistics. When the probability of an event depends on a posterior probability MCMC can be used to estimate the probability of the event. A Markov Chain exists when there exists a set of states $\{X_1, X_2, \ldots, X_n\}$ and the probability $P(X_n|X_{n-1})$ depends only on the state X_{n-1}, not any of the other previous states [6].

8 Summary

Monte Carlo simulations are an exciting, powerful tool for any numerics researcher due to their power and versatility. Numerics researchers should be pleased to find that it addresses one of the biggest challenges when it comes to using root finding methods. They should also consider using Monte Carlo methods when integrating functions, as it is a perfect way to counteract "the curse of dimensionality". Even looking beyond just numerics, researchers of all kinds have found Monte Carlo Methods extremely useful for their work. While it has been used for hundreds of years, advances in computing, simulations will only increase in ease and speed, meaning Monte Carlo simulations will only grow in importance as time goes on.

References

[1] W. F. Bauer. The monte carlo method. *Journal of the Society for Industrial and Applied Mathematics*, 6(4):438–451, 1958.

[2] P.L. Bonate. A brief introduction to monte carlo simulation. *Clinical Pharmacokinetics*, 40:15–22, 2001.

[3] Robert L. Harrison. Introduction to monte carlo simulation. In *AIP Conference Proceedings*, volume 1204. American Institute of Physics, 2010.

[4] Dirk P. Kroese, Tim Brereton, Thomas Taimre, and Zdravko I. Botev. Why the monte carlo method is so important.

[5] Vahid Mirzaei, Vahid Mirzaei Mahmoud Abadi, and Shila Bahnamriri. The approximate calculation of the roots of algebraic equation through monte carlo method. *International Journal of Mathematics and Computational Science*, 2:64–68, 08 2016.

[6] David Spiegelhalter W.R. Gilks, S. Richardson. *Markov Chain Monte Carlo in Practice*. 1996.