CS206P Project MC: Monte Carlo vs Deterministic Volume Integration

This report is organized as six sections. Section I introduces my program and how to use it. Section II elaborates on the techniques and algorithms I employ in my program to improve accuracy and efficiency. Section III, IV, and V display and analyze Part 1, 2, and 3's results, respectively. Finally, Section VI concludes insights gained from this project. For simplicity, I use **MC** to denote Monto Carlo integration and **CB** to denote Cube-based integration in this report. Primary tables in this report are placed in a separate file called "project_mc_tables.pdf" for easy reference.

Content

I. Introduction to My Program

My program consists of the following components:

- **makefile**: used to compile my program.
- **main.cpp**: main function and used to run each Part of the project. User can run the program by entering the project folder and typing "./project_mc" in the command line.
- **integration.h** and **integration.cpp**: define classes Integration, Monte_Carlo, and Cube_based. An object of the class represents a calculation of ball volume satisfying the given criteria.
- **timer.h** and **timer.cpp**: provided by the Professor Hirschberg as a tool to measure execution time.

My program is designed as follows:

At first, it asks which part you would like to run:

- Part 1: Given precision, α, and d, find the ball volume
- Part 2: Given N and d, find the ball volume
- Part 3: Given N and d, find the ball volume and use it to deduce π
- Part 4: unused
- Part 5: Given N and D, find the ball volume for d = 1, 2, ... , D
- Part 6: Given N and D, find the ball volume and use it to deduce π for d = 1, 2, ... , D

And then it will ask for parameters like precision, α, d, N, and D based on which part you choose. After you input all parameters, if you choose Part 1, 2, and 3, it will start to run MC. Since CB usually takes a "much" longer time, after MC is finished, it will ask you if you would like to continue with CB. If you answer yes, it will ask you whether you would like to run CB based on fixed precision or N because there is a huge time difference between the two criteria. The following figure shows an example of running Part 1:

Figure: Example of Running My Program

```
Please specify Part Number: (or input exit to quit)
1 2 3 4(unused) 5(auto run Part 2) 6(auto run Part 3)
1

Please specify precision in terms of digits (e.g., 4 means tol = 0.001): (or input exit to quit)
4

Please specify alpha (%) for Monte Carlo Method: (or input exit to quit)
0.1 0.2 0.5 1 2 5 10 20
1

Please specify dimension (d): (or input exit to quit)
6

Part 1: Given precision = 4 and alpha = 1%, find 6-ball volume

Monte Carlo starts to generate result...
[Attempt 1] p = 0.090(0.023), N = 4.348064e+08
Estimated time: 18 minute(s)
d = 6, N = 4.348064e+08, M = 10: answer = 5.167(0.001), interval = [5.166, 5.168]

The result of Monte Carlo:
The answer is 5.167(0.001) when N = 4.348064e+08 and M = 10
The true value is within [5.166, 5.168] at 99.0% confidence level
The number of seconds of run time: 3095
The correct answer is 5.168 and the diffs is 5.816962e-04

Do you want to continue with Cube-based? (May need to wait for a lifetime): (or input exit to quit)
Yes: input 1, No: input 0
1

Please specify precision in terms of digits (e.g., 4 means tol = 0.001)
or input 0 if you would like to specify N instead: (or input exit to quit)
0

Please specify N (input 0 for default value of 1 million): (or input exit to quit)
1000000000000
```

If you choose Part 5 or 6, my program will calculate Part 3 or 4 for d = 1, 2, ... , D.

After each run, my program will print out detailed information of each calculation like answer, error, lower bound, upper bound, and how many sample points/cubes are used to perform the calculation (N). My program will also print out a table summarizing all numbers so that you can copy it to a text file or a spreadsheet for subsequent processing. The following figure is a sample output of MC:

Figure: Sample Output of MC

```
Part 1: Given precision = 4 and alpha = 1%, find 6-ball volume

Monte Carlo starts to generate result...
[Attempt 1] p = 0.090(0.023), N = 4.348064e+08
Estimated time: 18 minute(s)
d = 6, N = 4.348064e+08, M = 10: answer = 5.167(0.001), interval = [5.166, 5.168]

The result of Monte Carlo:
The answer is 5.167(0.001) when N = 4.348064e+08 and M = 10
The true value is within [5.166, 5.168] at 99.0% confidence level
The number of seconds of run time: 3095
The correct answer is 5.168 and the diffs is 5.816962e-04
```

Other features of my program include:

- When it starts to perform calculation, it will give a ballpark of how long you may have to wait based on the method you choose and the sample size N.
- If N is quite large (e.g., N = 10^{12}), it will show the progress of the calculation so that you know that your computer is working hard instead of sleeping. The following two figures show examples of MC and CB, respectively. Please note that in MC's case, each thread will print out its own progress, while in CB's case, the main thread collects all threads' information and print out the overall progress. Please refer to "Multi-threading" in the next section for more details.

Figure: Sample Output of MC in Progress

```
Monte Carlo starts to generate result...
[Attempt 1] p = 0.006(0.006), N = 8.410099e+08
Estimated time: 1 hour(s)
[Random Check (per thread)] i = 1.000000e+08 (%24, elapsed time: 1650)
[Random Check (per thread)] i = 1.000000e+08 (%24, elapsed time: 1651)
[Random Check (per thread)] i = 1.000000e+08 (%24, elapsed time: 1652)
[Random Check (per thread)] i = 1.000000e+08 (%24, elapsed time: 1655)
[Random Check (per thread)] i = 1.000000e+08 (%24, elapsed time: 1656)
[Random Check (per thread)] i = 1.000000e+08 (%24, elapsed time: 1658)
[Random Check (per thread)] i = 1.000000e+08 (%24, elapsed time: 1659)
[Random Check (per thread)] i = 1.000000e+08 (%24, elapsed time: 1667)
```

Figure: Sample Output of CB in Progress

```
Cube-based starts to generate result...
Estimated time: 14 hour(s)
[Random Check (all threads)] k = 8 (%8, elapsed time: 12947)
[Random Check (all threads)] k = 16 (%16, elapsed time: 19312)
[Random Check (all threads)] k = 24 (%24, elapsed time: 25652)
```

II. Techniques and Algorithms Employed in My Program

I use the following techniques and algorithms to improve the accuracy and efficiency of my calculation:

Monte Carlo Integration

Minimize Number of Runs

Let the number of runs be M. Then the total number of sample points is N*M. According to the Law of Large Numbers, we know that as the total sample size increases, our answer will approach the true answer, and the precision improves as well. We can expand the total sample size by raising either N or M. In consideration that M has a higher overhead (e.g., calculating average and standard deviation), I minimize M to be 10 and focus on increasing N to reduce error.

Estimate N Using Binomial Distribution

In Part 1, MC will decide N based on user's input and increase N until the specified precision is reached (i.e., half of confidence interval is less than the tolerance). Below is my algorithm to decide N.

Let v_i be a sample point among the sample of size N. Let B_i be a random variable which is 1 if v_i is inside the ball and 0 if v_i is outside the ball. Then $\{\mathrm{Bi}\}_{i=1}^{N}$ is a set of i.i.d random variables which follows the Bernoulli distribution with p = proportion of ball volume to cube volume. Denote the number of sample points inside the ball by *I*. Then $\mathrm{I} = \sum_{i=1}^{N} B_i$ follows a binomial distribution B(N, p). Therefore, variance of *I* is Np(1-p). It implies that if we know p, then we can deduce how large N should be such that the error is less than the tolerance. The following shows the deduction process:

Notation:

- N: number of sample points in a run
- p: proportion of ball volume to cube volume, an unknown constant
- I: number of sample points inside the hypersphere, a random variable
- C: cube volume, a known constant = 2^d
- S: ball volume in a run, a random variable = $\frac{I}{N} * C$
- M: number of runs
- t: t-value based on the given α
- σ(X): standard deviation of a random variable X
- s: sample standard deviation of ball volume in M runs
- tol: tolerance based on the given precision
- error: half of confidence interval based on the given α

$\mathrm{I} \sim \mathrm{B}(\mathrm{N}, \mathrm{p})$

$\Rightarrow \sigma(I) = \sqrt{Np(1-p)}$

$$\text{error} = \mathrm{t} * \frac{s}{\sqrt{M}} \approx \mathrm{t} * \frac{\sigma(S)}{\sqrt{M}} = t * \frac{\sigma\left(\frac{I}{N} * C\right)}{\sqrt{M}} = t * \frac{\frac{C}{N}\sigma(I)}{\sqrt{M}} = \mathrm{t} * \frac{\frac{C}{N}\sqrt{Np(1-p)}}{\sqrt{M}}$$

$$\text{tol} > \text{error} \approx \mathrm{t} * \frac{\frac{C}{N}\sqrt{Np(1-p)}}{\sqrt{M}}$$

$$\Rightarrow \mathrm{N} > \mathrm{t}^2 * \frac{C^2 * p(1-p)}{M * tol^2}$$

In order to make the error less than the prespecified tolerance, I choose $\mathrm{N} = \mathrm{t}^2 * \frac{C^2 * p(1-p)}{M * tol^2}$ as the sample size.

How should we know p? In order to estimate p, MC has a trial run using a sample of size $N' = \frac{1}{tol}$, and then use the following formula to calculate p:

$$\hat{p} = \frac{I}{N'} = \frac{\sum_{i=1}^{N'} B_i}{N'}$$

Besides, I use the following formula to calculate p's error:

$$p\_error = \mathrm{z} * \frac{\sqrt{\hat{p}(1-\hat{p})}}{\sqrt{N'}}$$

where z = 2.58

According to the Central Limit Theorem, when N is large, $\hat{p} = \frac{\sum_{i=1}^{N'} B_i}{N'}$ is close to normal distribution N(p, $\frac{\mathrm{p}(1-\mathrm{p})}{N'}$), so here I use z-value (set $\alpha = 1\%$) instead of t-value.

In order to get a conservative N so that we do not have to make a lot of attempts, I pick the point within p's confidence interval which is closest to 0.5 to calculate N. (N has a maximum at p = 0.5.)

However, in the case of bad luck, although $\hat{p}$ may have been quite close to p and give a reasonable estimate of N, error may keep being higher than the tolerance and the loop never breaks. To avoid this situation, I multiple N by $\left(\frac{error}{tol}\right)^2$ and try again.

According to the formula $\mathrm{N} = \mathrm{t}^2 * \frac{C^2 * p(1-p)}{M * tol^2} = \mathrm{t}^2 * \frac{2^{2d} * p(1-p)}{M * tol^2}$, it is worth noticing that when d is larger, C is larger and so could N. However, at the same time, p becomes farther away from 0.5 and p(1-p) becomes smaller (please refer to Table 0), which somewhat offsets the increase in C^2. It explains why the required sample size of MC to obtain a specified precision does not exponentially increase with d.

I choose N as an even number to facilitate antithesis variates, which is described below.

Generate Sample Using Antithetic Variates

In MC, a sample point is a vector of d elements. Since each element X_k follows uniform distribution over the range [-1, 1], V = (X_1, X_2, .. , X_d) has the same distribution as AV = (-X_1, -X_2, .. , -X_d). It suggests that the expected value of *I* will be the same in both samples. Therefore, after generating a sample point v, I negative each element of v to form an antithetic sample av and add it into the sample pool. In this way, I can get the equally good estimate of volume while reducing its variance.

Calculate Standard Deviation and Average Using Pipelining

I use the algorithm below to calculate the squared sum of errors and the average on the fly:

Let S_j be the observed ball volume of jth run, where j = 1, 2, ... , M. Let Q_j, A_j be the squared sum of errors, average of the first j runs, respectively. Then,

For j = 1, 2, ... , M

$$\mathrm{Q}_j = Q_{j-1} + \frac{j-1}{j}(S_j - A_{j-1})^2$$

$$\mathrm{A}_j = A_{j-1} + \frac{S_j - A_{j-1}}{j}$$

Denote the sample standard deviation by *s*. Using Q_M we can get *s* = $\sqrt{\frac{Q_M}{M-1}}$.

The method allows us to continuously update the squared sum of errors and the average without the need to save observation data. Another advantage is that the rounding errors are reduced.

Note: Iterator in my program starts from 0, so the formula in my program looks a little different from above.

Source: https://en.wikipedia.org/wiki/Standard_deviation

Bonus: Confidence Interval

As mentioned above, when N is large, $\hat{p} = \frac{\sum_{i=1}^{N'} B_i}{N'}$ is close to normal distribution, and so is S = $\hat{p}$*C. Therefore, Sj obtained from jth run can be considered an observation from a normal distribution and the average $\frac{\sum_{j=1}^{M} S_j}{M}$ should follow Student's t-distribution. It means that we can construct the exact ball volume's confidence interval using t-value.

When my program starts, it automatically builds a hash table of α against t-value. Since M = 10, the hash table is based on degree of freedom = 10 – 1 = 9. When user inputs α, my program looks up the hash table and assign the corresponding t-value.

After the average and the sample standard deviation (*s*) are calculated, the confidence interval is computed as below:

standard error = $\frac{s}{\sqrt{M}}$
error = t-value * standard error
upper bound = average + error
lower bound = average - error

We can say that the interval [lower bound, upper bound] contains the true value at confidence level (1 – α).

Cube-based Integration

Estimate K Using Surface Ratio

In Part 1, CB will decide N based on user's input and increase N until the specified precision is reached (i.e., half of intersection cube volume is less than the tolerance). Below is my algorithm to decide K.

Let p be the proportion of the ball's surface area to the cube's surface area. Let X be the number of intersection cubes (which intersect the ball's surface) and F be the number of cubes which are located at the surface of the enclosing hypercube. I assume that $\hat{p} = \frac{X}{F}$ will be close to p when K is large. The following shows how to use p to derive K such that the error is less than the tolerance:

volume of X = X*$\left(\frac{2}{K}\right)^d$
Since $F = K^{d-1} * d * 2$ and $X \approx p * F$,
=> error = half of volume of X = 0.5*X*$\left(\frac{2}{K}\right)^d \approx 0.5 * p * F * \left(\frac{2}{K}\right)^d = \cancel{0.5} * p * K^{d-1} * d * \cancel{2} * \left(\frac{2}{K}\right)^d$
$= p * \frac{\cancel{K^d}}{K} * d * \frac{2^d}{\cancel{K^d}} = \frac{p * d * 2^d}{K}$

$$\text{tol} > \text{error} \approx \frac{p * d * 2^d}{K}$$
$$\Rightarrow \text{K} > \frac{p * d * 2^d}{tol}$$

As with MC, in order to estimate p, CB also has one trial run using $K' = \frac{1}{tol}$. To be conservative, I multiply $\hat{p}$ by $1 + p\_error$, which is a heuristic error term equal to $\frac{1}{K'}$. So the estimation formula of K is:

$$\text{K} = \frac{(1 + p\_error) * \hat{p} * d * 2^d}{tol}$$

I choose K as an even number to maintain the symmetry of small cubes (i.e., the number of cubes on the positive side is equal to the number of cubes on the negative side in each dimension).

The following table summarizes the estimation formulas of error and sample size for the two methods:

Table: Estimation Formulas of Error and Sample Size for MC and CB

	Input	Error	Sample Size
MC	p: proportion of ball volume to cube volume, estimated by $\hat{p} = \frac{I}{N'} = \frac{\sum_{i=1}^{N'} B_i}{N'}$ might be adjusted by $p\_error = \text{z} * \frac{\sqrt{\hat{p}(1-\hat{p})}}{\sqrt{N'}}$ where z = 2.58	error $\approx \text{t} * \frac{\frac{C}{N}\sqrt{Np(1-p)}}{\sqrt{M}} = O(\frac{1}{\sqrt{N}})$ Source: statistical error	$\text{N} = \text{t}^2 * \frac{C^2 * \hat{p}(1-\hat{p})}{M * tol^2}$ $= \text{t}^2 * \frac{2^{2d} * \hat{p}(1-\hat{p})}{M * tol^2} = O(4^d)$
CB	p: proportion of the ball's surface area to the cube's surface area, estimated by $\hat{\text{p}} = \frac{X}{F}$ adjusted by $p\_error = \frac{1}{K'}$	error $\approx \frac{p*d*2^d}{K} = \frac{p*d*2^d}{N^{\frac{1}{d}}} = O(\frac{1}{N^{\frac{1}{d}}})$ Source: use discrete cubes to estimate continuous ball volume	$\text{K} = \frac{(1 + p\_error) * \hat{p} * d * 2^d}{tol}$ $\text{N} = K^d = (\frac{(1+p\_error)*\hat{p}*d*2^d}{tol})^{\boldsymbol{d}} =$ $\text{O}((cd2^d)^d)$, where c = $\frac{(1+p\_error)*\hat{p}}{tol}$

The table shows why CB needs "much" larger N in order to achieve the same precision as MC as d increases. When d = 1, MC's error is of the order of $\frac{1}{\sqrt{N}}$, which is larger than CB's error, which is of the order of $\frac{1}{N}$, so CB is preferred. When d = 2, both errors are of the same order of $\frac{1}{\sqrt{N}}$. When d ≥ 3, CB's error is larger than MC's error given the same N. That is why CB needs larger N to reduce its error to the given tolerance.

The formulas of sample size corroborate the fact. When d increments by 1, MC's sample size grows with a multiple of 4 (which can be offset by decrease in $p(1-p)$), but CB's sample size grows with a multiple of whopping $cd2^d$.

Surprisingly, even though p's definitions are different in MC and CB's formulas, they are exactly the same values! To see why, let's list the formulas of volume and surface area of n-ball first: (R: radius of the ball, $\Gamma(x)$: Gamma function)

n-ball's volume: $S_n = \frac{\pi^{\frac{n}{2}} R^n}{\Gamma(1+\frac{n}{2})}$ ... (1)

n-ball's surface area: $A_n^S = \frac{n\pi^{\frac{n}{2}} R^{n-1}}{\Gamma(1+\frac{n}{2})}$ ... (2)

Source: http://scipp.ucsc.edu/~haber/ph116A/volume_11.pdf

And here is the formulas of volume and surface area of n-cube:
n-cube's volume: $C_n = (2R)^n$ ... (3)
n-cube's surface area: $A_n^C = (2R)^{n-1} * n * 2$ ... (4)

Dividing (1) by (3) and (2) by (4) we get:

MC's p = $\frac{Ball\ Volume}{Cube\ Volume} = \frac{S_n}{C_n} = \frac{\frac{\pi^{\frac{n}{2}} R^n}{\Gamma(1+\frac{n}{2})}}{(2R)^n} = \frac{\frac{\pi^{\frac{n}{2}} \cancel{R^n}}{\Gamma(1+\frac{n}{2})}}{2^n * \cancel{R^n}} = \frac{\pi^{\frac{n}{2}}}{2^n * \Gamma(1+\frac{n}{2})}$

CB's p = $\frac{Ball's\ Surface\ Area}{Cube's\ Surface\ Area} = \frac{A_n^S}{A_n^C} = \frac{\frac{n\pi^{\frac{n}{2}} R^{n-1}}{\Gamma(1+\frac{n}{2})}}{(2R)^{n-1} * n * 2} = \frac{\frac{\cancel{n}\pi^{\frac{n}{2}} \cancel{R^{n-1}}}{\Gamma(1+\frac{n}{2})}}{2^{n-1} * \cancel{R^{n-1}} * \cancel{n} * 2} = \frac{\pi^{\frac{n}{2}}}{2^n * \Gamma(1+\frac{n}{2})}$

The two formulas are exactly the same, so MC's p is equivalent to CB's p. However, my program still pretends that it does not know the secret; MC and CB still estimate their own p based on the definitions separately.

Table 0 in "project_mc_tables.pdf" shows the estimated N based on the formulas above. The table utilizes the convenient fact above and calculate MC and CB's N using the same p. The attached figure in this table suggests that MC's N increases smoothly with d, as opposed to the explosive growth of CB's N (up until d = 17). The table also estimates the required time ("Est Time (s)" in terms of seconds and "Est Time (h)" in terms of hours) if we assume that a computer can handle 10^6 sample points for MC and 10^7 sample cubes for CB per second. It seems that the time required to calculate 3-ball volume using CB (55 hours) is even more than the time required to calculate 20-ball volume using MC (40 hours)!

Query All Cubes Using DFS Tree

In order to systematically and efficiently sample every small cube, I construct a DFS tree, where each leaf node represents a sample cube. A node is an array of d+2 elements. A base point of a cube is the edge point which has the smallest sum of elements. Let v = (x_1, x_2, ... , x_d) be the base point of the cube. Let half of the number of segments be K' = $\frac{K}{2}$ and segment size be h = $\frac{2}{K}$. Then the domain for each x_k = {-K'h, -(K'-1)h, ... , 0, ... , (K'-1)h}. node[k] saves x_k, k = 1, 2, ... , d.[1] Incrementing one or multiple elements of the base point of the cube by h generates an edge point of the cube. node[0] saves the smallest distance between the cube's edge point and the origin, and node[d+1] saves the largest distance. The following figure illustrates a node's structure:

Figure: Node's Structure in CB

node[0]	node[1]	node[2]	...	node[d]	node[d+1]
the smallest distance between the cube's edge point and the origin	x_1	x_2		x_d	the largest distance between the cube's edge point and the origin

$x_k \in$ {-K'h, -(K'-1)h, ... , 0, ... , (K'-1)h}, k = 1, 2, ... , d

[1] In order to avoid rounding error, node[k], k = 1, 2, ... , d, actually saves the "ordinal" of x_k, and is translated to actual value when we calculate the distance. For example, if x_1 = -K'h, then node[1] = 0; if x_1 = 0, then node[1] = K'. The relationship between x_k and node[k] is x_k = -1 + h*node[k].

The following figure is a simplified representation of our tree. A node in Dimension i represents a possible combination of the first i elements of a base point of a cube, and each child in Dimension i inherits the first (i-1) elements of its parent. Because the domain for each element in a vector has K distinct possible values, -K'h, -(K'-1)h, ... , 0, ... , (K'-1)h, each parent node has K children.

We start from x_1 = -K'h and then explore its smallest child x_2 = -K'h. We continue to explore the smallest child of each node. During this process, we keep updating node[0] and node[d+1]. Finally, we come to the node in Dimension d where each of its elements is -K'h. (The first path is highlighted in yellow.)

We use the function Is_Inside() to decide if it is inside the ball. Since we have saved both the smallest edge distance and the largest edge distance in the node, the task is quite straightforward. If the smallest edge distance is more than 1, then the cube is completely outside the ball; if the largest edge distance is less than or equal to 1, then the cube is completely inside the ball; otherwise, it intersects the ball's interface.

Once (x_1, x_2, ... , x_d) = (-K'h, -K'h, ... , -K'h) is explored, we continue to explore its siblings with the same x_1, x_2, ... , x_{d-1} but different x_d values. They are highlighted in green in the following figure. Because they inherit the same parent (x_1, x_2, ... , x_{d-1}) = (-K'h, -K'h, ... , -K'h), they can use their parent's information plus their individual x_d to construct their nodes and decide their relative locations to the ball. Once all children have been explored, we continue to explore the children of their parent's siblings (highlighted in blue in the following figure). Once all siblings' children have been explored, we continue to explore children of the siblings of the siblings' parent. In this way, we can sample all small cubes within the hypercube and determine how many cubes are inside, outside, and intersecting the ball.

Figure: DFS Tree in CB

Dimension										
1					-K'h	-(K'-1)h ...	0	... (K'-2)h	(K'-1)h	
2			-K'h	-(K'-1)h ...	0 ...	(K'-1)h				
3		-K'h	-(K'-1)h	... (K'-1)h						
...										
d-1	-K'h	-(K'-1)h ...	(K'-1)h							
d	-K'h -(K'-1)h ...	(K'-1)h								

Estimate Volume Using Intersection Ratio

Instead of assuming that half of the intersection cube is within the ball, we can use the information of the smallest edge distance and the largest edge distance to estimate how many percent of the intersection cube is actually within the ball. The estimation formula is as follows:

$$p' = \frac{1 - \text{smallest edge distance}}{\text{largest edge distance} - \text{smallest edge distance}}$$

If the intercept is close to the edge point with the smallest distance from the origin, we assume that only a small part of the cube is within the ball and should give a discount on the volume of the cube; on the contrary, if the intercept is far away from the edge point with the smallest distance from the origin, we assume that a big part of the cube is within the ball and should give a premium on the volume of the cube. We multiply each intersection cube's volume by its own p' and then add it to the sum of the volume of the cubes inside the ball to estimate the ball volume. Let I, X be the number of cubes inside, intersecting the ball, respectively. The ball volume (S) can be represented as the following formula:

$$S = I * h^d + (\sum_{i=1}^{X} p_i' * h^d)$$

Common

Multi-threading

Since the andromeda machines have 8 cores each, my program divides the work to 8 threads. Each has its own memory space to save results so that they will not compete.

- Monte-Carlo: Each thread performs a run of generating N sample points and estimating the ball volume based on the sample.
- Cube-based: Each thread expands a parent node whose first element is -K'h, -(K'-1)h, ... , 0, ... , (K'-1)h.

Calculate Exact Answer Using Recursive Function

The close-form formula of the n-ball volume is as follows: (R: radius of the ball)

$$V_{2k}(R) = \frac{\pi^k}{k!}R^{2k},$$
$$V_{2k+1}(R) = \frac{2^{k+1}\pi^k}{(2k+1)!!}R^{2k+1} = \frac{2(k!)(4\pi)^k}{(2k+1)!}R^{2k+1}.$$

Source: https://en.wikipedia.org/wiki/Volume_of_an_n-ball

Instead of directly applying the formula, I use the recursion method to reduce the rounding errors and avoid the overflow problem. (Please refer to line 64 to 73 in integration.h.)

III. Part 1's Result and Analysis

Part 1 requires us to calculate ball volume using MC and CB given 4 digits of precision. For MC, I apply $\alpha = 1\%$, which is equivalent to confidence level of 99%. Because when $d \geq 3$, the estimated time for CB may be more than two days, I fix N to be 10^{12} instead. Table 1 of "project_mc_tables.pdf". shows the result for Part 1.

Which method is more efficient as d becomes large, given that we insist on 4 digits of precision in the answer?

Answer:

MC can generate a value to 4 digits of precision within a reasonable time frame (e.g., 7 hours when d = 7). However, when $d \geq 3$, CB becomes intractable. Therefore, MC is more efficient as d becomes large.

How far can you push d for each method?

Answer:

According to Table 0, for MC, theoretically I can push d to 16 given time limit of one day. However, for CB, if we would like to "guarantee" 4 digits of precision, I can only push d to 2 because when $d \geq 3$, the time required is more than 2 days. For comparison purposes, when $d \geq 3$, I fix $N = 10^{12}$ when using CB because the workload seems to be able to be finished within one day. Due to time constraint, Table 1 only shows both methods' results up to d = 8.

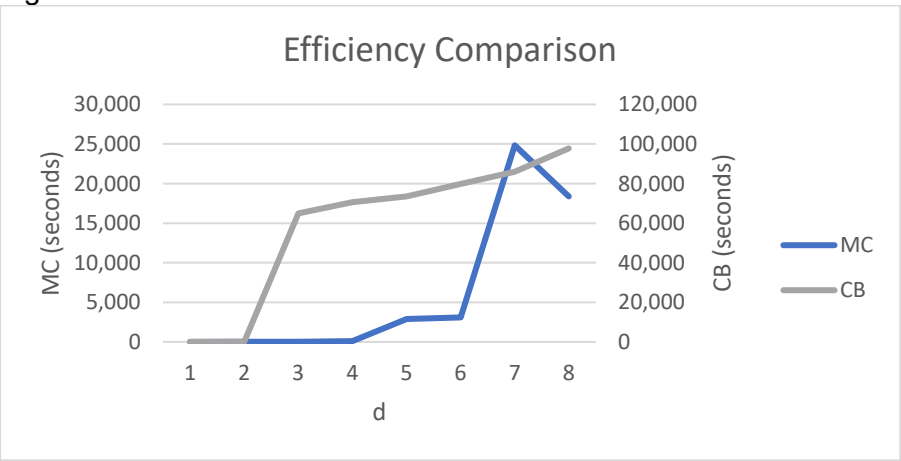
Plot figures showing run times and accuracies as a function of d.

Answer:

According to Table 1, given the same precision, CB can give a more accurate answer. The reason is that there is some stochasticity in MC, so we can only say the true value is within the interval at some confidence level. However, when $d \geq 3$, CB takes a much longer time even though we do not require its precision. Because I fix N instead of precision for CB, when $d \geq 5$, MC gives more accurate answers.

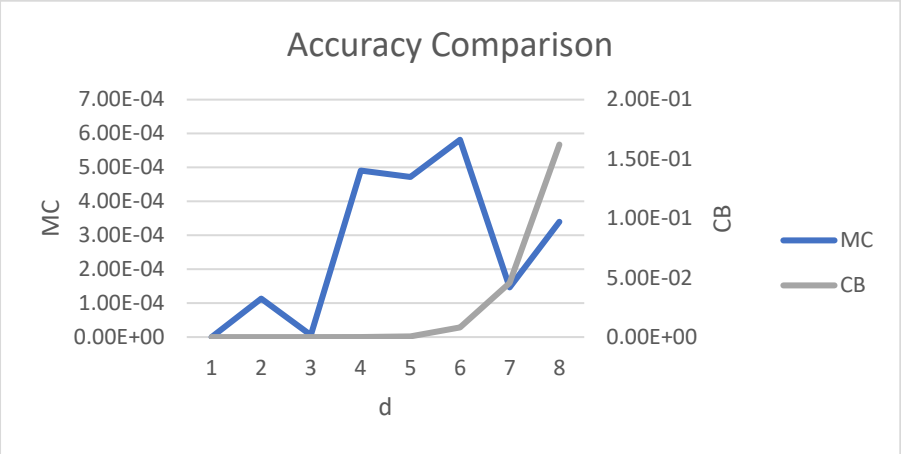
The following figure shows MC and CB's run time as a function of d. Please note that the left axis is for MC and the right axis is for CB. Clearly, both require more time as d increases. Even though I fix N, CB's run time still increases with N. That is because when d increases, the height of the DFS tree also increases, and there are more parent nodes in the tree. MC's run time jumps at d = 7, while CB's run time jumps at d = 3. According to the metrics of the axis, it is apparent that MC is more efficient than CB.

Figure: MC and CB's Run Time as a Function of d



The following figure shows MC and CB's accuracy as a function of d. Please note that the left axis is for MC and the right axis is for CB. Here accuracy is calculated as the difference between the estimated answer and the exact answer obtained from the close-form formula mentioned in the last section. Because we require 4 digits of precision for MC with high confidence (99%), its accuracy can also be maintained under 10^{-3} = 0.001. Because I fix N instead of precision for CB when d ≥ 3, its accuracy quickly reduces as d increases.

Figure: MC and CB's Accuracy as a Function of d



How do things change if we demand 8 digits of precision?

Answer:

According to estimation formulas of sample size, N is proportional to tol^{-2} for MC and tol^{-d} for CB. If we demand 4 digits of precision, then tol = 10^{-3} and N's multiple is 10^{6} for MC and 10^{3d} for CB. If we demand 8 digits of precision instead, then tol = 10^{-7} and N's multiple is 10^{14} for MC and 10^{7d} for CB, which means 10^{8} and 10^{4d} times of increase in N for MC and CB, respectively. In this case, both methods become intractable.

Table: Change in N when Changing Precision from 4 to 8

Method	N's multiple\Precision	4 (tol = 10^{-3})	8 (tol = 10^{-7})	Change in N
MC	tol^{-2}	10^{6}	10^{14}	10^{8}
CB	tol^{-d}	10^{3d}	10^{7d}	10^{4d}

IV. Part 2's Result and Analysis

Part 2 requires us to fix N = 10^6 and compare MC and CB's results, which are shown in Table 2 of "project_mc_tables.pdf".

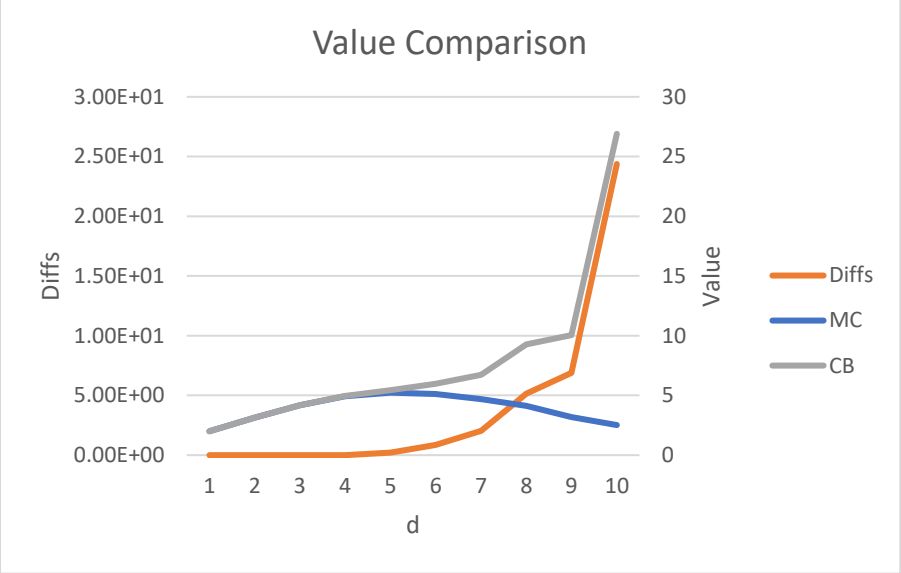
Show the difference between the two values as a function of d.

Answer:

The following figure shows MC and CB's values and their differences as a function of d. Please note that the left axis is for value difference and the right axis is for values. As we can see from the figure, the difference exponentially increases with d. Besides, MC's value is consistently lower than CB's value.

According to Table 0, we know that the ball volume peaks at d = 5. However, CB's value strictly increases with d, which is incorrect. On the contrary, MC's value also peaks at d = 5, which coincides with the exact value.

Figure: MC and CB's Values and Their Difference as a Function of d



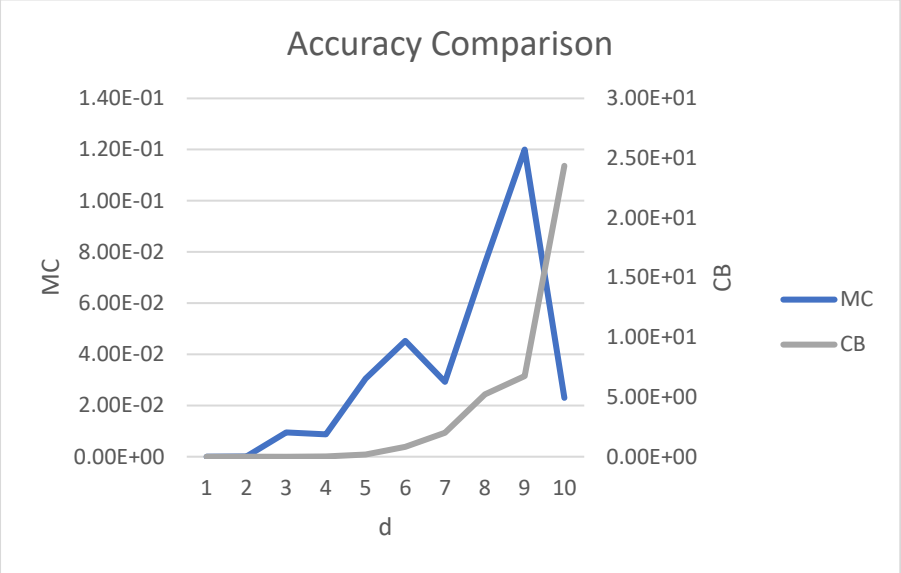
Based on Part 1, which do you think is giving the more accurate answer?

Answer:

According to Table 2, we can see that CB gives more accurate values only when d = 2 and 3. When d ≥ 4, MC gives more accurate values. Therefore, if we have limited CPU cost and time, CB is the more preferred method to solve volume in low dimensions, while MC is more useful in the case of high dimensions.

The following figure shows MC and CB's accuracy as a function of d. Please note that the left axis is for MC and the right axis is for CB. When N is fixed, there is no apparent pattern between MC's accuracy and d. On the contrary, CB's accuracy consistently reduces when d increases. That is because increase in d will cause $K = N^{\frac{1}{d}}$ to decrease, reducing the granularity of the segmentation and thus accuracy.

Figure: MC and CB's Accuracy as a Function of d



V. Part 3's Result and Analysis

Part 3 requires us to calculate π with different combination of (N, d, method), where N = 100, 100^2, 100^3, 100^4, d = 2 to 10, and method = MC or CB. The method is calculating ball volume first, and then reversing the close-form formula to solve π. Table 3 of "project_mc_tables.pdf" shows the results. Because when d = 1, ball volume is independent of π, I neglect this case.

If I give you a fixed value of N (number of samples), what is the most accurate value (including minimizing the size of the uncertainty interval) that you can compute for π? In other words, which method, and what value (or values) of d should you use to compute π as accurately as possible if you have only a specified number of samples at your disposal, and you don't know N in advance?

Answer:

According to Table 3, we can see that if we only have a fixed sample size, the best choice is to use CB to calculate 2-ball volume, and then reverse the formula to solve for π.

The following table shows the most accurate value of π as a function of N. Each value is generated from calculating 2-ball volume using CB.

Table: Most Accurate Value of π as a Function of N

N	100	100^2	100^3	100^4
π (true value: 3.141593)	3.123506	3.141395	3.141591	3.141593
Accuracy	1.81E-02	1.98E-04	2.06E-06	2.08E-08

The reasons could be the following:

- When d = 2, we don't need to take any root to deduce π; actually, the volume is exactly π. The less mathematical operations are involved, the smaller error the answer should have.
- CB's value is deterministic, so we can be sure that the answer is close to the true value; on the contrary, MC's value is stochastic, and it is likely that the answer is far away from the true value.

Like Part 2, given d, MC gives more accurate values of π when d ≥ 4 in the case of N = 100, 100^2, 100^3. When d is small, CB is still the better choice.

The following figures show MC and CB's accuracy as a function of d when N = 100, 100^2, 100^3, 100^4, respectively. Please note that the left axis is for MC and the right axis is for CB.

Figure: MC and CB's Accuracy of π Value as a Function of d when N = 100

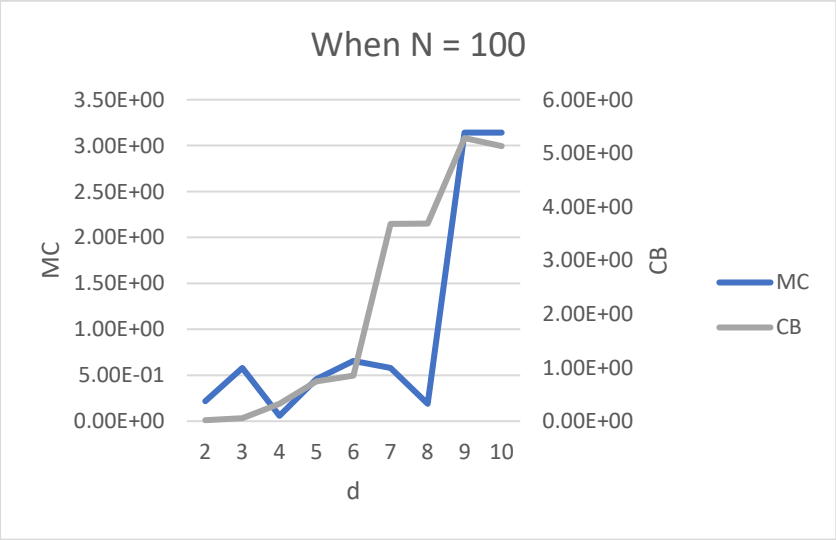


Figure: MC and CB's Accuracy of π Value as a Function of d when N = 100^2

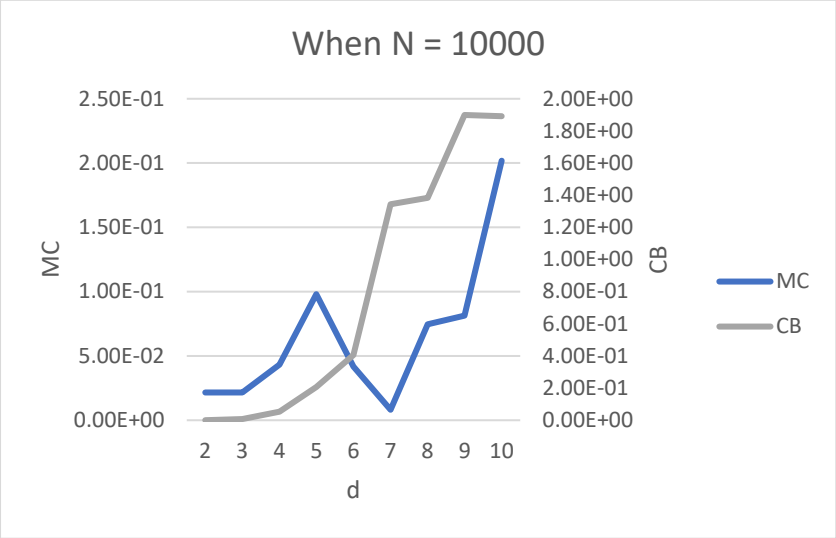


Figure: MC and CB's Accuracy of π Value as a Function of d when N = 100^3

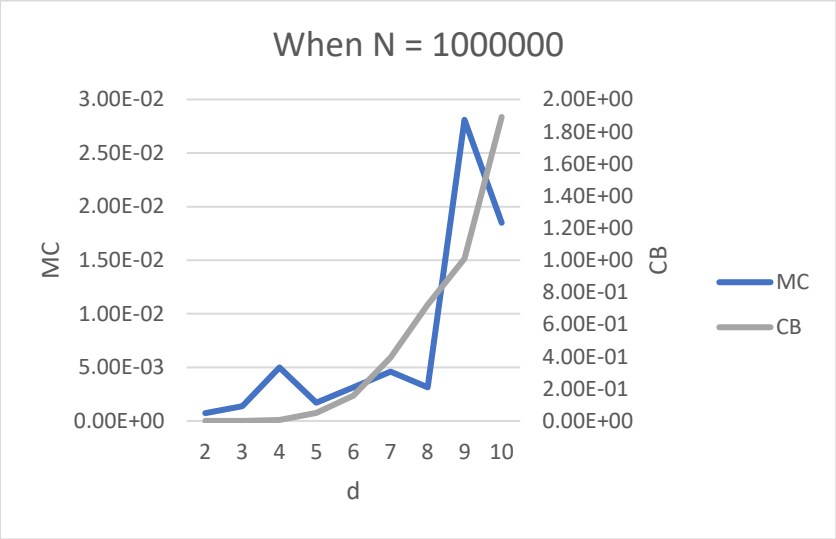
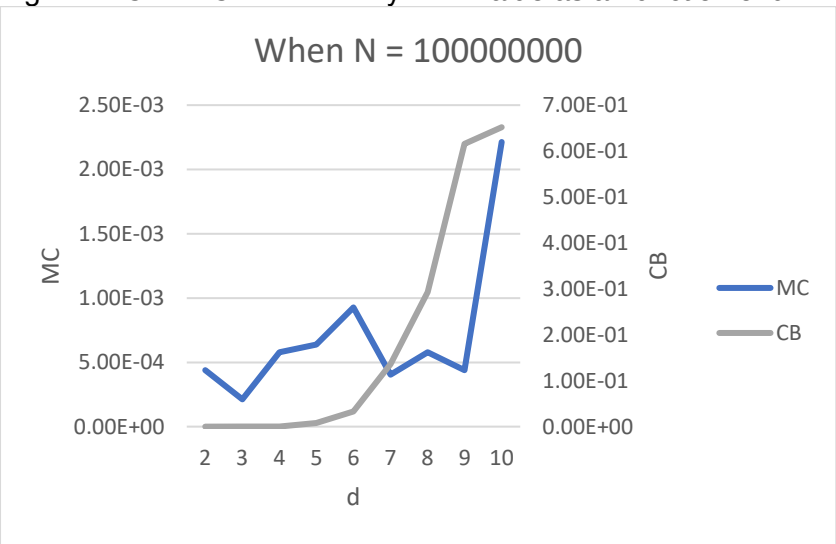


Figure: MC and CB's Accuracy of π Value as a Function of d when N = 100^4



VI. Conclusion

The following table summarizes the pros and cons of each method.

Table: Pros and Cons of MC and CB

	MC	CB
Advantage	When $d \geq 3$, MC is the only viable method to ensure a specified high precision	Give a deterministic answer and can guarantee that the true value is within some range
Disadvantage	Stochastic, i.e. cannot guarantee the specified precision in each attempt; sometimes require many attempts to reach the specified precision, and it could be possible that the specified precision will never be reached (in the case of bad, bad luck)	When $d \geq 3$, CB becomes impractical if a high precision is required

According to the results, the project gives us the following implication: If we would like to estimate the volume of a multi-dimensional object, CB will give a more accurate answer. However, due to the "curse of dimensionality", as dimension increases, it becomes impossible to apply CB given limited time. In this case, only MC can solve for the answer. Therefore, the general rule is: solve low-dimensional problem using CB, and solve high-dimensional problems using MC.