CS 206P Project Fall 2018
Monte Carlo vs Deterministic Volume Integration

KARTHIK VELLANKI 5720552
NIHAL NILESH GANDHI 58404096
RATHNAINTHRI HANNAH DEVANAND 64207299

Content:

Introduction:

The focus of this project is to compare the Monte Carlo integration and Deterministic Volume integration for estimating the volume of a d-dimensional hypersphere. For both the methods, a d-dimensional hypersphere of radius 1, centered at the origin is surrounded by a d-dimensional hypercube with sides of length 2 and centered at the origin.

Monte Carlo Method:

The function nSphereVolume( ) takes N random points (coordinates of d-dimensions) and the counter count_in_sphere keeps track of how many points are within a distance of 1. All the points within a distance of 1 from the origin would be within the sphere. (count_in_sphere ÷ N) * Volume of hypercube = Estimate of the volume of the hypersphere

The function runs( ) takes an input for the dimension of the hypersphere and the number of random points and executes the function nSphereVolume( ) until a precision of 10^{-4} is reached.

Precision is calculated using the following formula:
(Mean + 2(standard deviation) – (Mean – 2*(standard deviation))) ÷ Mean*
The formula can be simplified to:
4(standard deviation) ÷ Mean*

The function time( ) determines how long it takes to reach a precision of 10^{-4} using the Monte Carlo method.

The Monte Carlo method has a modular function isWithinSphere(d), so that it can be replaced by another module function that deals with another d-dimensional shape.

Deterministic Volume Integration (Cube-Based method):

Each side of the surrounding hypercube is sliced into ‘K’ pieces, which means the hypercube is sliced into ‘h’ small hypercubes with sides of length (2 ÷ K). The volume of each of the small hypercubes is $(2 \div K)^d$

Each of the small hypercubes is either wholly within a distance of 1 from the origin, wholly outside a distance of 1 or only partially inside a distance of 1. The lower bound is defined as the total volume of all the cubes within a distance of 1. The middle bound is defined as the total volume of all the cubes that are only partially within a distance of 1 from the origin. The upper bound is the sum of the lower bound and the middle bound. To determine how many small hypercubes are entirely within a distance of 1, we determine how many small hypercubes have their outermost vertex within a distance of 1 from the origin. The lower bound counter keeps track of how many small hypercubes are entirely within a distance of 1.

Likewise, to determine how many small hypercubes are only partially within a distance of 1 from the origin, we determine how many small hypercubes have their innermost vertex within a distance of 1 **and** outermost vertex outside a distance of 1 from the origin. The middle bound counter keeps track of the number of small hypercubes that are only partially within a distance of 1 from the origin.

‘n’ is the position number of each of the small hypercubes. Since each side of the surrounding hypercube is sliced into ‘K’ segments, ‘n’ takes values from 1 to K÷2 (considering only the positive direction from the origin(meaning the first quadrant) – final values are multiplied by 2^d to determine the volume values).

For a specific dimension, we loop through all possible combinations of ‘n’ (1 <= n <= (K ÷ 2)) in all the dimension positions to find the position numbers of the outermost vertex of each small hypercube. Each of the tuples is then multiplied by 2 ÷ K (since the small hypercube has the length of 2 ÷ K) to find the final coordinates of the outermost vertex. If the outermost vertex is determined to be outside a distance of 1, we then proceed to check whether the innermost vertex is within a distance of 1 (to add it to the middle bound). The position numbers of the innermost vertex are all possible combinations of

n-1 (1<= n <= (K ÷ 2)) in all the dimension positions. Again, each of the position numbers is multiplied by 2 ÷ K to find the final coordinates of the innermost vertex.
(lower_bound ÷ total) * (2^d) = lower bound of the volume of the hypersphere
((lower_bound + middle_bound) ÷ total) * (2^d) = upper bound of the volume of the hypersphere

We iterate through different values of 'K' till we reach a precision of 10^{-4}.
Precision in the cube based method is the difference between and the upper bound and the lower bound of the volume.

The Cube-based method has a modular function check_cube_inside(dist), so that it can be replaced by another module function that deals with another d-dimensional shape.
Throughout this project, we are calculating accuracy using this formula:

Accuracy= 100-((Difference between Actual Volume and Calculated volume)/ Actual Volume)*100

Part 1:

In the Monte Carlo method, we were able to push 'd' up to 15 dimensions with 4 digits of precision.For the 16th dimension in the first run we see 0, hence we called out 15 as our stopping point for Monte-Carlo. Whereas for cube based the time taken for dimension=4 was greater than the time taken for dimension 15 for Monte-Carlo.

As 'd' becomes large the Monte Carlo method becomes more efficient compared to the cube-based integration method. The run times for the cube-based method become unreasonably large for higher dimensions. We start with k=1000 and iterate till 4 digits of precision is maintained.

Achieving 8 digits of precision would not be reasonable for our experiment because the run times become extremely large for the Monte Carlo method and Cube-based method.

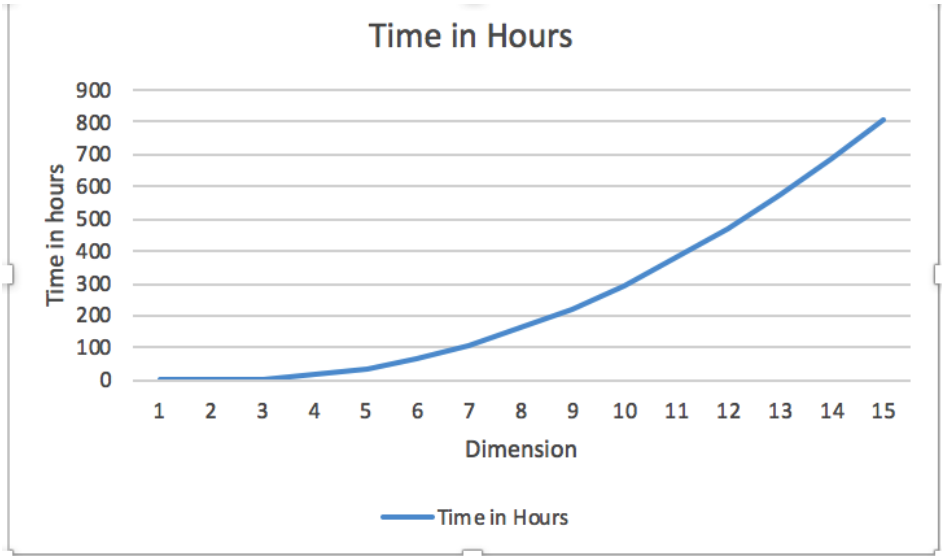
The plots of runtime vs d, accuracy for every dimension are displayed below. We observed a bell curve trend for every plot of dimension vs the population of volumes. The dimensions until 7 are plotted here as an example.

Dimensions	Actual Volume	Monte Carlo Volume	N	m	MC Run Time (s)	MC Run Time (h)	MC Accuracy(%)	Cube Based Volume	k	CB Run Time (s)	CB Run Time (h)	CB Accuracy(%)
1	2	2	1,000,000	1	11	0.003055556	100	1.999996	20000	11	0.003055556	99.9998
2	3.141592654	3.141583446	1,000,000	419	3684	1.023333333	99.99970692	3.13977053	63000	18183	5.050833333	99.942
3	4.188790205	4.188919611	1,000,000	1439	9513	2.6425	99.99691066	4.186000471	126000	584132	162.2588889	99.9334
4	4.934802201	4.934959043	1,000,000	3359	51489	14.3025	99.99682172	4.930410227	198000	1697858	471.6272222	99.911
5	5.263789014	5.26400377	1,000,000	6179	129612	36.00333333	99.99592012					
6	5.16771278	5.168071882	1,000,000	9899	243882	67.745	99.99305105					
7	4.72476597	4.724665871	1,000,000	14519	394299	109.5275	99.9978814					
8	4.058712126	4.058829478	1,000,000	20039	580863	161.3508333	99.99710866					
9	3.298508903	3.298682579	1,000,000	26459	803574	223.215	99.99473469					
10	2.55016404	2.549971464	1,000,000	33779	1062432	295.12	99.9924485					
11	1.884103879	1.884113419	1,000,000	41999	1357437	377.0658333	99.99949365					
12	1.335262769	1.335239951	1,000,000	51119	1688589	469.0525	99.99829114					
13	0.910628755	0.910662382	1,000,000	61139	2055888	571.08	99.99630721					
14	0.599264453	0.59928508	1,000,000	72059	2459334	683.1483333	99.99655783					
15	0.381443281	0.381409293	1,000,000	83879	2898927	805.2575	99.99108965					

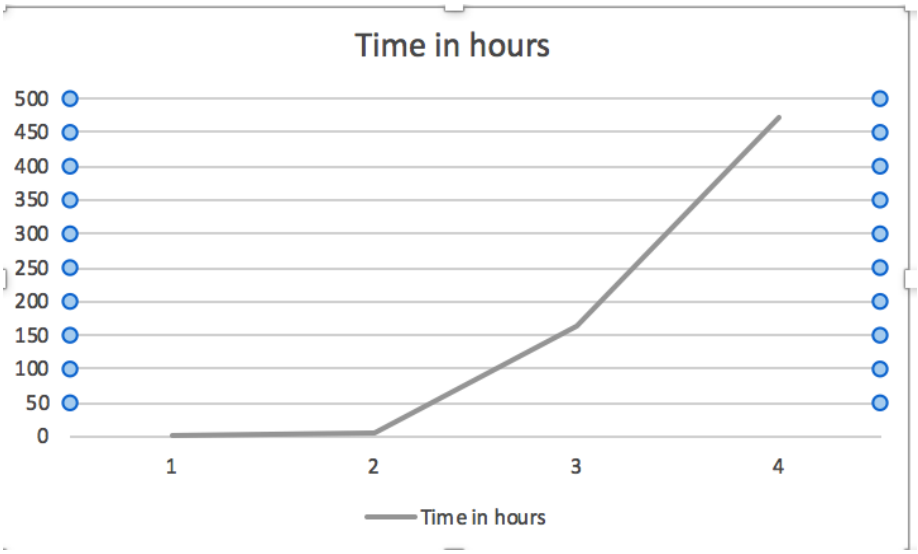
Here,
N = Number of random points taken within the hypercube for a single run
m = Number of runs for a particular dimension
k = Number of slices on each side of the dimension

This table summarizes the various volumes obtained using Cube-Based and Monte-Carlo with their accuracies and runtime where precision is constant across all dimensions.

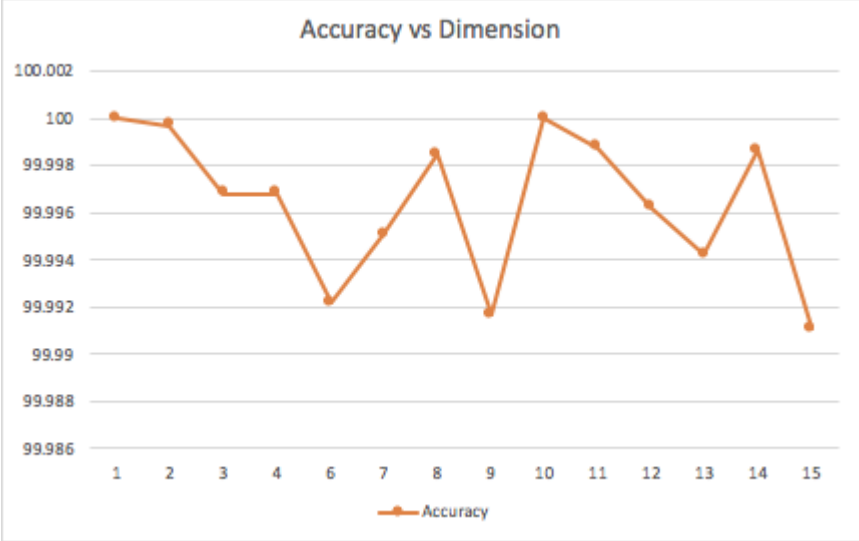
Runtime vs Dimension in Monte-Carlo:



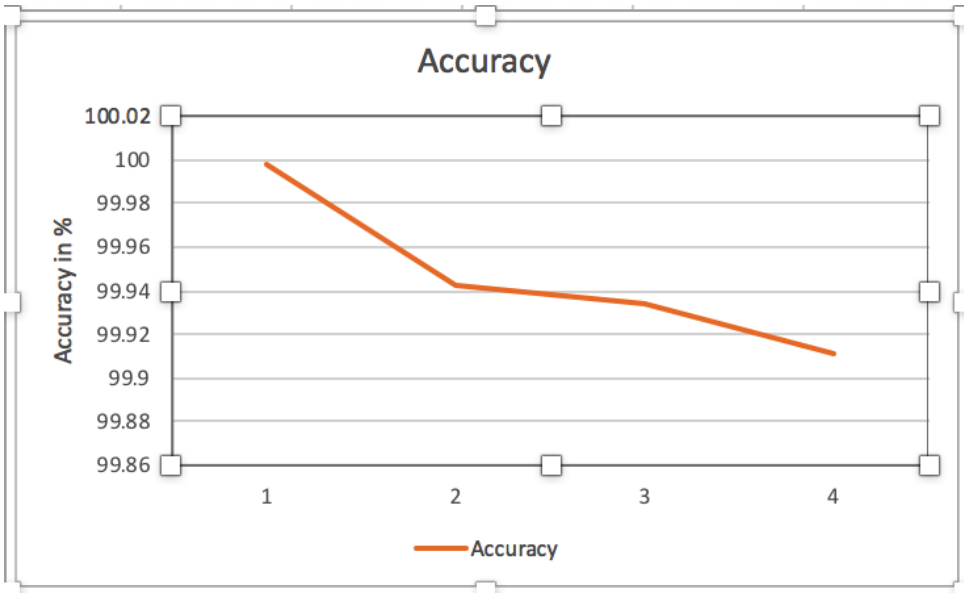
Runtime vs Dimension in Cube based:



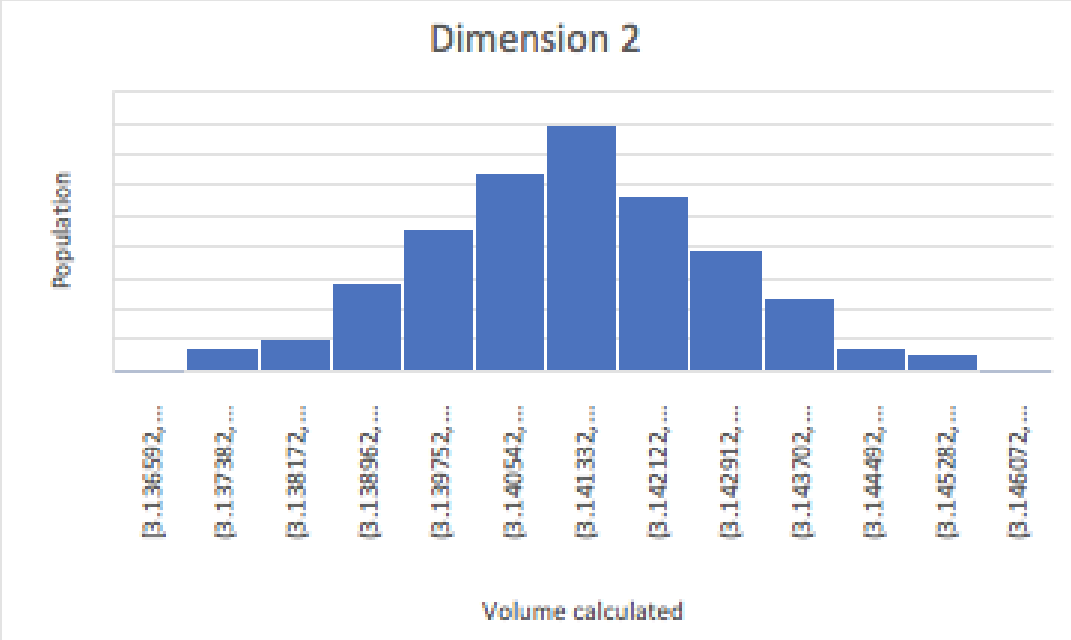
Accuracy vs Dimension for Montecarlo:

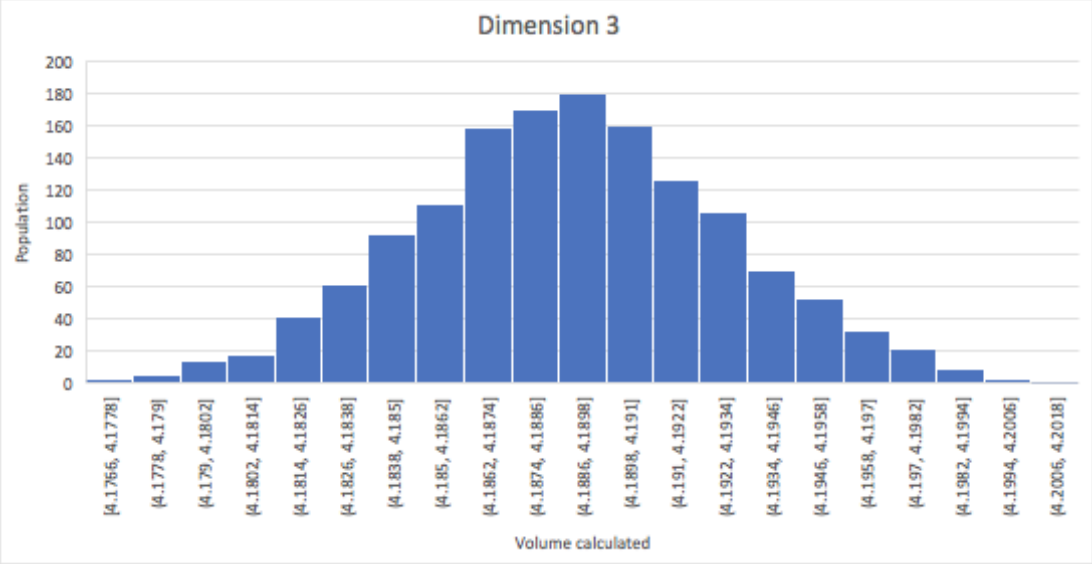


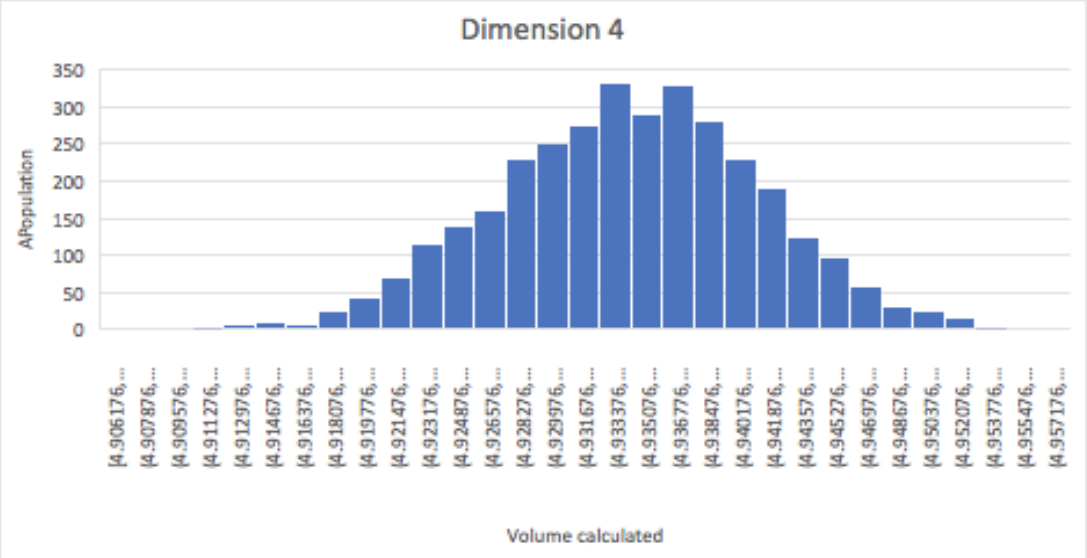
Accuracy vs dimension in Cube based:

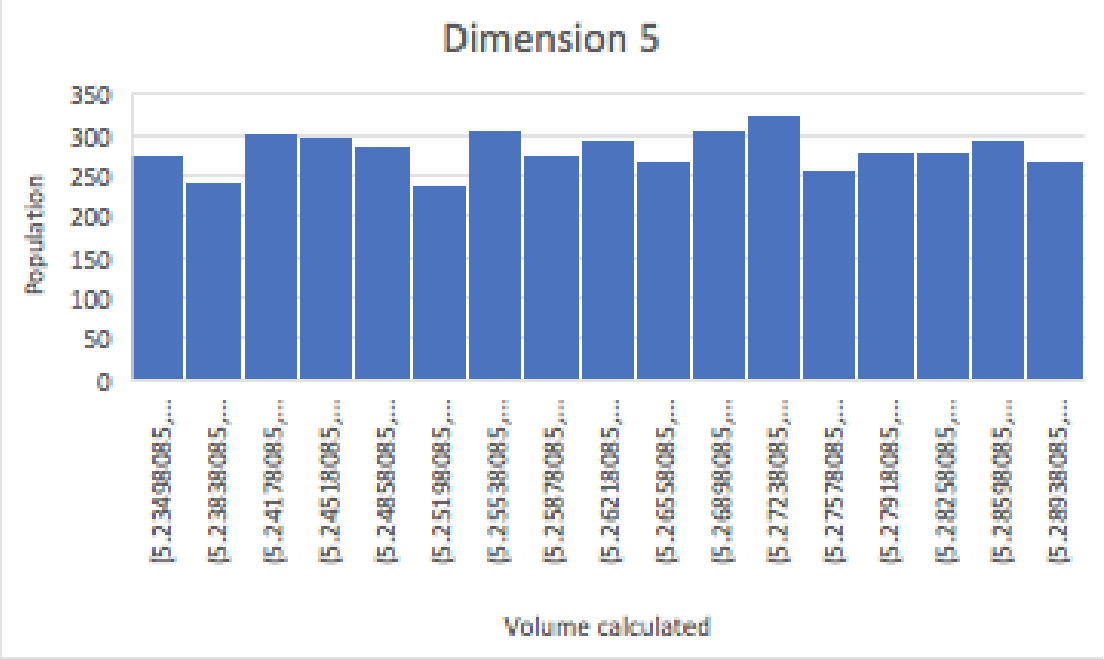


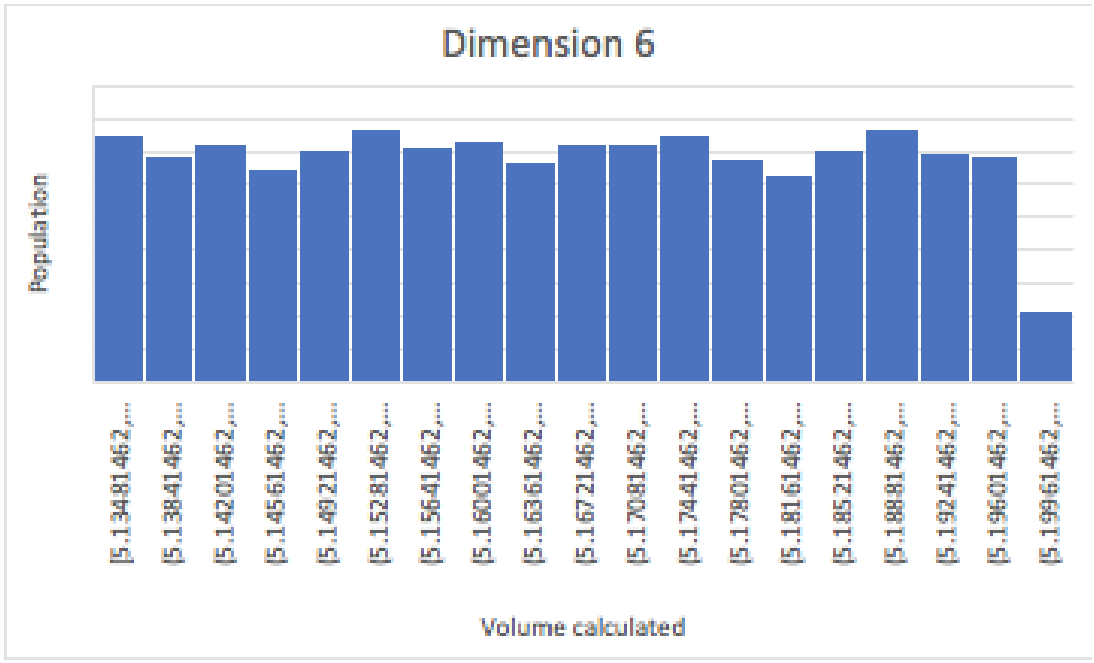
Histogram per dimension (Plotting volumes vs Population)

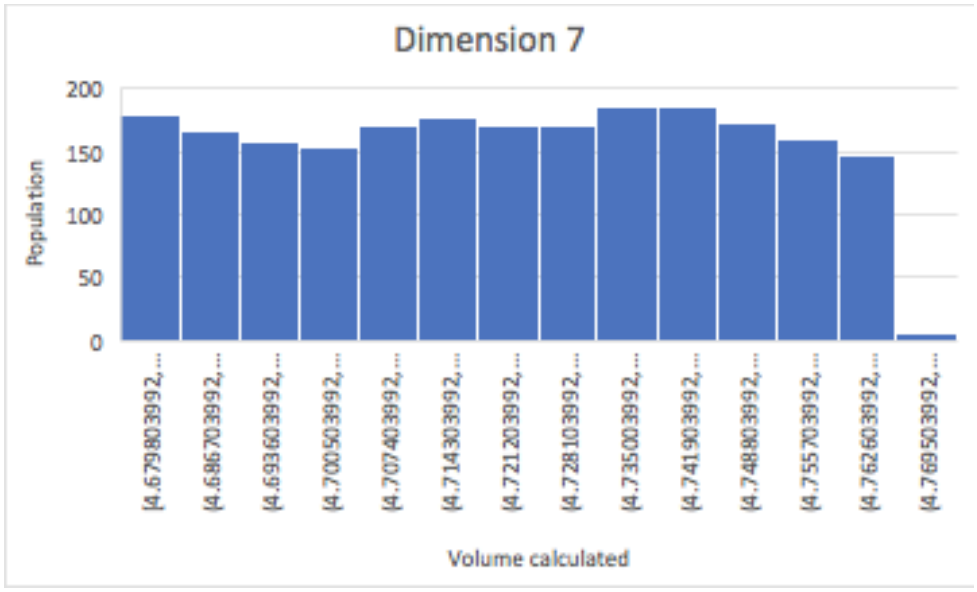





Dimension 4
APopulation
350
300
250
200
150
100
50
0
[4.906176,...
[4.907876,...
[4.909576,...
[4.911276,...
[4.912976,...
[4.914676,...
[4.916376,...
[4.918076,...
[4.919776,...
[4.921476,...
[4.923176,...
[4.924876,...
[4.926576,...
[4.928276,...
[4.929976,...
[4.931676,...
[4.933376,...
[4.935076,...
[4.936776,...
[4.938476,...
[4.940176,...
[4.941876,...
[4.943576,...
[4.945276,...
[4.946976,...
[4.948676,...
[4.950376,...
[4.952076,...
[4.953776,...
[4.955476,...
[4.957176,...
Volume calculated


Dimension 5
Population
350
300
250
200
150
100
50
0
[5.2349808 5,...
[5.2383808 5,...
[5.2417808 5,...
[5.2451808 5,...
[5.2485808 5,...
[5.2519808 5,...
[5.2553808 5,...
[5.2587808 5,...
[5.2621808 5,...
[5.2655808 5,...
[5.2689808 5,...
[5.2723808 5,...
[5.2757808 5,...
[5.2791808 5,...
[5.2825808 5,...
[5.2859808 5,...
[5.2893808 5,...
Volume calculated


Dimension 6
Population
(5.134814 62,...
(5.138414 62,...
(5.142014 62,...
(5.145614 62,...
(5.149214 62,...
(5.152814 62,...
(5.156414 62,...
(5.160014 62,...
(5.163614 62,...
(5.167214 62,...
(5.170814 62,...
(5.174414 62,...
(5.178014 62,...
(5.181614 62,...
(5.185214 62,...
(5.188814 62,...
(5.192414 62,...
(5.196014 62,...
(5.199614 62,...
Volume calculated


Dimension 7
Population
200
150
100
50
0
(4.679803992,...
(4.686703992,...
(4.693603992,...
(4.700503992,...
(4.707403992,...
(4.714303992,...
(4.721203992,...
(4.728103992,...
(4.735003992,...
(4.741903992,...
(4.748803992,...
(4.755703992,...
(4.762603992,...
(4.769503992,...
Volume calculated

Observations:
As per the observation from the graphs, we see that the time increases as the dimensions increase. As we are trying to maintain fixed accuracy (4 digits precision) , we are compromising on the runtime.

Part 2:

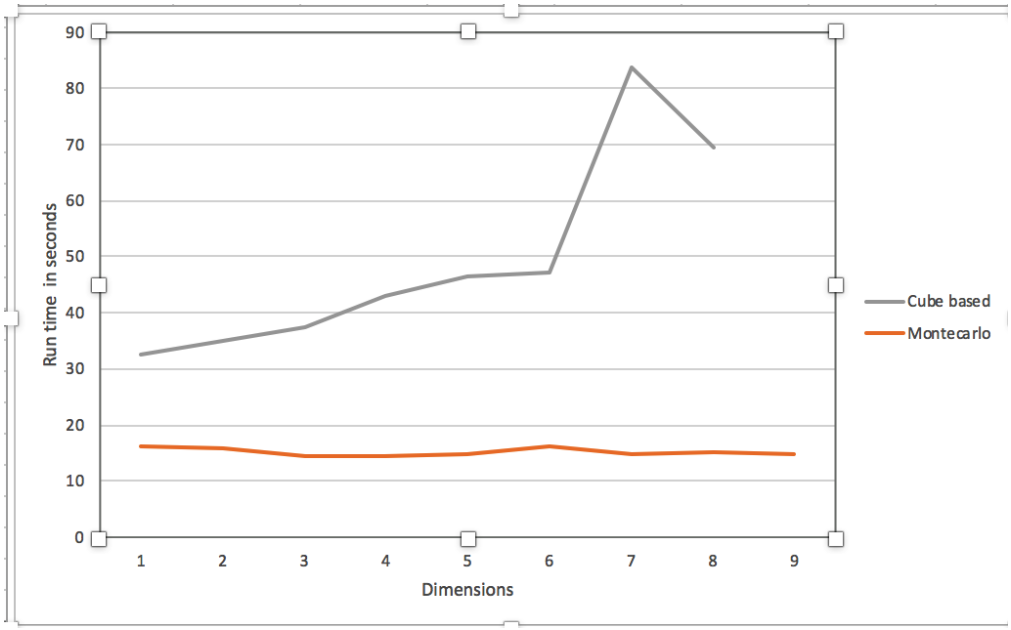
In the cube-based method, the volume of the hypersphere is assumed to be the lower bound of the volume + half of the middle bound of the hypersphere. As the number of dimensions increases, the number of hypercubes decreases. This leads to time being less in higher dimensions for cube based & the accuracy decreases.In this part, we increment the dimension until any of the volumes reached using either of the methods is zero. From our experiment, the volume hit 0 at dimension 9.
K at each step is k=N ^ (1/d)

N = 1,000,000

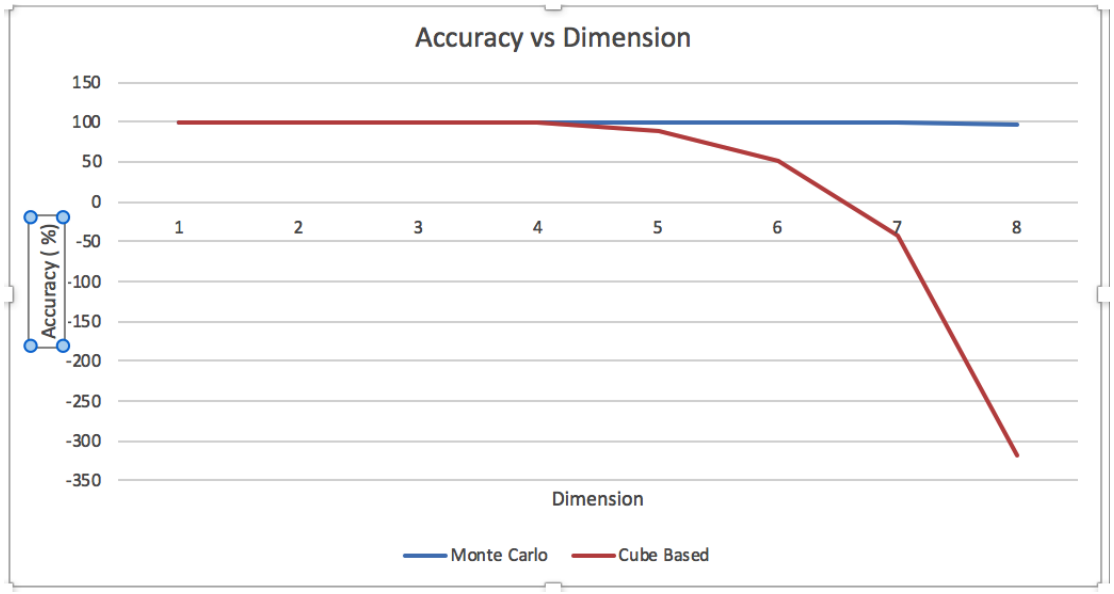
Dimensions	Actual Volume	Monte Carlo Volume	MC Run Time (s)	MC Accuracy(%)	Cube Based Volu	k = N^(1/d)	CB Run Time (s)	CB Accuracy(%)	Winner
1	2	2	16.10391903	100	2	1000000	16.32322907	100	PAR
2	3.141592654	3.144164	15.89143705	99.9181515	3.141288	1000	18.99596715	99.99030258	CB
3	4.188790205	4.188192	14.45562291	99.98571891	4.187616	100	23.06649709	99.97196793	MC
4	4.934802201	4.921232	14.56784105	99.72501024	5.01184082	32	28.46005607	98.43887117	MC
5	5.263789014	5.258656	14.7913599	99.90248443	5.848144531	16	31.68607616	88.89857637	MC
6	5.16771278	5.161856	16.21204996	99.88666591	7.749632	10	31.1271019	50.03748602	MC
7	4.72476597	4.735872	14.61092019	99.76494011	11.44140625	8	69.08810091	-42.15815814	MC
8	4.058712126	4.110336	15.280375	98.72807255	21.01143118	6	54.83126187	-317.6871512	MC
9	3.298508903	3.257856	14.76107812	98.76753697	233	5	8.867474079	-6863.797821	MC

This table summarizes the various volumes obtained using Cube-Based and Monte-Carlo with their accuracies and runtime where N is constant and k is calculated from N to make it comparable.

Runtime vs Dimension for Montecarlo & Cube based:



Accuracy vs Dimension for Montecarlo & Cube based:



Observation:

1. As per the experiment's instruction, we are keeping N and k comparable , therefore Monte-Carlo is faster as the dimensions increase.
2.As per the experiment's instruction, we are keeping N and k comparable, therefore Monte-Carlo is more accurate as the dimensions increase.

Part 3:

The analytical formula for the volume of a n-dimensional hypersphere is.

$$V_n(R) = \frac{\pi^{\frac{n}{2}}}{\Gamma\left(\frac{n}{2}+1\right)} R^n,$$

(Referenced from Wikipedia)
To accurately compute the value of π we compare MC and CB for N =1000,10000,100000,1000000 for varying values of d. Since we are using a hypersphere of radius 1, we can backtrace the above formula to find π.
For every value and N the 'Winner" changes. The observations for various N and d is plotted below:

N = 1000

Dimensions	Exact Answer	MC method	CB method	MC Accuracy (%)	CB Accuracy (%)	Winner
1	3.141592654	3.141592654	3.13531261	100	99.8001	MC
2	3.141592654	3.136	3.1015625	99.82198031	98.90186543	MC
3	3.141592654	3.111123554	3.095024552	99.03013845	99.48253413	CB
4	3.141592654	3.12409987	3.413842555	99.44318742	90.72556268	MC
5	3.141592654	3.27846771	4.510324869	95.64313164	62.42582609	MC

N = 10,000

Dimensions	Exact Answer	MC method	CB method	MC Accuracy (%)	CB Accuracy (%)	Winner
1	3.141592654	3.141592654	3.140964366	100	99.980001	MC
2	3.141592654	3.1308	3.1304	99.65645917	99.98722371	CB
3	3.141592654	3.146395716	3.144867622	99.84711378	99.95143351	CB
4	3.141592654	3.142483095	3.243701589	99.97165638	96.77902819	MC
5	3.141592654	3.127077835	4.69265433	99.53797896	49.93484082	MC
6	3.141592654	3.080150664	5.798889998	98.04424074	11.73356016	MC
7	3.141592654	3.150986822	5.426410413	99.70097432	27.78695313	MC

N = 100,000

Dimensions	Exact Answer	MC method	CB method	MC Accuracy (%)	CB Accuracy (%)	Winner
1	3.141592654	3.141592654	3.141529822	100	99.99800001	MC
2	3.141592654	3.14196	3.160951771	99.988307	99.39554385	MC
3	3.141592654	3.132030271	3.277673261	99.69561991	95.34988561	MC
4	3.141592654	3.141464627	3.183940804	99.99592477	98.64788618	MC
5	3.141592654	3.147974031	4.099684784	99.79687445	69.76751575	MC
6	3.141592654	3.14100229	5.213371323	99.98120814	34.02204642	MC
7	3.141592654	3.149537815	4.590749958	99.74709766	54.24051946	MC
8	3.141592654	3.162469909	7.378524944	99.33545632	-33.31526169	MC

N = 1,000,000

Dimensions	Exact Answer	MC method	CB method	MC Accuracy (%)	CB Accuracy (%)	Winner
1	3.141592654	3.141592654	3.14158637	100	99.9998	MC
2	3.141592654	3.140248	3.141288	99.95719835	99.9668816	CB
3	3.141592654	3.14058547	3.141005524	99.96794037	99.986625	CB
4	3.141592654	3.145210963	3.166019842	99.88482564	99.3383948	MC
5	3.141592654	3.148796421	3.276707855	99.77069695	95.93776743	MC
6	3.141592654	3.138288382	3.595926256	99.89482178	85.41759654	MC
7	3.141592654	3.140881563	4.044757361	99.97736529	71.22222601	MC
8	3.141592654	3.140832249	4.738781881	99.97579557	49.12336905	MC

Observation:

1. There is no clear winner when d is varied for each N value.

Conclusion:

From the plots and the data, it is evident that the Monte Carlo method is more efficient to compute the value of a d-dimensional hypersphere for higher dimensions. The run times for the cube-based method becomes unreasonably large for higher dimensions. The cube-based method gives us definite bounds for the volume of the hypersphere in lower dimensions.

Additionally, one of the disadvantages with the Monte Carlo method is the fact that the reaching precision requires an unpredictable number of runs because we are taking random points within the hypercube.