Personal Information

Name: Bicheng Wang
StudentID: 64444637
Name: Chenyang Ren
StudentID: 22002644
date:August 31, 2018

Work Division

Because if following the basic idea of Cube-base, we cannot test more than dimension 3, we developed several different method on Cube-Base. And with the discussion, we generate a now method modified from the evenly divide space idea of Cube-Base and the distribution idea from Monte Carlo to deal with the general require PI or require $Hypersphere volume$ problem fast and beyond the dimension limitation. The result is that we can use this modified method to calculate even to 64 dimension with 4 digits percision only by 6443 second.

Monte Carlo method developed by Chenyang Ren, test and analyse by both, improved by both; Cube-Base basic method 1 developed by Chenyang Ren, Cube-Base method 2 developed by Bicheng Wang, analyse by both; Cube-Base now modified method develop by Bicheng Wang, improve and analyse by Both.

Method Description

Answer function

According to the table on Wikipedia. We can use the hypercube formula in the code to compare with our result. The answer method is in code 004.

Monte Carlo General method

The code is at java package $MonteCarloNCubeBase$. The Monte Carlo Method developed at both class $MonteCarloNCubeBase$ and $MonteCarloNCubeBase1TestPart2$.

Basic Idea

For each sample, generate d random numbers in the range [0,1] as its 1st, 2nd,..., d dimensions, add the square of these values as the square of the distance to the origin. if If the sum is greater than 1, this sample is outside the hypersphere, if the value is less than 1, then it is inside the hypersphere. The number of samples in the hypersphere plus half of that on the hypersphere divided by the total number of samples is the volume of the hypersphere.

Find Nd for each d using loop test

The basic idea of finding Nd is to start the test with the smallest possible value and see if it meets the requirements. If it does not meet the requirements, expand the value and continue testing until we find a suitable value.

Specifically, since it requires 4 digit precision, the minimum number of samples is at least 1000×2^d. When $d = 2$, the minimum value is 4000.

Then, we check if the difference between the result generated by this sample number and the correct result is greater than 0.001. If it is greater, we multiply the number of samples by 10 and repeat the test; If the error is smaller than 0.001, the sample number may be an appropriate Nd.

Cube based Integration method

We use two edition of Cube-Base methods to deal with the Cube-Base problem. The first is at java package $MonteCarloNCubeBase$. The second is at java package $CubeBase2$.

The First is just consider if the sample is in the hyper sphere, part inside, or not. And consider the part inside is half weight to count into the total weight. The reason is the surface of hyper sphere is alway lower one dimension than hyper sphere volume. Once the sample scale become large, considering the hyper volume, the surface is a minor factor, so we just ignore them.

Since it requires 4-bit precision, its minimum number of samples is at least $\sqrt{4000}$=64. For each small hypercube, we take the closest distance from the hypercube to the origin. The specific implementation is using the recursive function. Each layer is one-dimensional. First, each recursive function calculates K, the length of the small hypercube. Then, it uses the for-loop, which uses 1/K, 2/K, ..., 1 respectively denotes the point whose the dimension coordinates are 1/K, 2/K, ... 1. Then, this function is called again as the next dimension.

The Second is that can we divide then hyper sphere by 2^d part every time. Count the inside part as current part weight into the total volume, count the outside part as zero contribution in total volume, and only consider the partially inside part to divide them further. After pruning some no need calculate sample, we can improve the performance of Cube-Base method.

Modified Method from Basic Cube-Base

The code is at java package $OurModifiedMethod$. The main idea is as Cube-Base evenly divide lower dimension space and calculate higher dimension sample distribution with their frequency by lower dimension.

The question can be convert to a subquestion that as follows:

$$x_1^2 + x_2^2 + ... + x_n^2 \leq R^2$$

Thus, we can just check if x_i is less than R^2 or not to estimate the hyper space volume.

The method is base on basic idea of above part to calculate high dimension volume. Because as the dimension d increase, to make sure the basic percision, our sample N also need to increase as the order of 2^d. But can we modify a little by remember the freqency or distribution character in each dimension level about the above method to generate a more efficient method without knowing Pi or $hypercubefunction$ before.

The first performance improvement is of course using multi-thread to parallel our program, we developed as $MultiThreadCal$ package; the second improvement is using Hadoop as an distributiong system to accelerate the parallel calculation, we haven't finished it yes; the third improvement is using the hypercube sample with its frequncy or said distribution condition.

Instead of just only generate hypercube sample with every sample frequncy is one, we generate sample with their own frequncy to compress the calculate work. Like, if we generate $x1, x2, x3$, we can also plus one frequncy count for other sample $x1, x3, x2$, $x2, x1, x3$, $x2, x3, x1$, $x3, x1, x2$, $x3, x2, x1$. By this way, one sample can be generate as $d!$ times. And according the hypercube defination function $x1^2 + x2^2 + ... + xn^2 < 1$, we found these sample has symmetry character. Thus, we can further compress these $d!$ sample to one sample with frequncy.

The formal method deducts from dimension 1, and assume there are N sample in dimension 1 with frequncy 1. The sample can either generate by Monte Carlo Integration method randomly or Cube-based Integration method evenly in every gap.

Then, to derive higher dimension sample with their frequncy, as the above said, we can join one $n/2$ sample with its frequncy with all other $n/2$ sample to generate n sample with new frequncy, and decide if this new sample is going into the n hyper-cube or not.(indeed the idea is from database joint, ha...) We run the method $generate()$. Every time, if we know the sample with their distribution in x dimension, we always can derive the distribution of

sample in $2x$ dimension by the join method in *generate*(). By this way, we first can derive $n/2$ dimension.

And the last step from $n/2$ to n, we use a *doubleDimension*() method to accelerate the last step calculate. Because we don't need know n dimension sample distribution any more, but we only care about how many sample is in the n hyper-cube. Thus, we just using *doubleDimension*() to directly fill in the frequncy of every $n/2$ point joint with other points can or cannot going into the n dimension.

Modified Method applied with Cube-based Integration method

Recommend method. In code, you can directly run $CubeBasedTest()$ in $Test$ class. The Cube-based Integration directly generate N sample in dimension 1.

Part 1

Monte Carlo Error Estimate

The result of Monte Carlo Method is randomly distributed in possible space, so we need to estimate the confidence interval of result. As we know that if using n sample to test, the sample error follows student's T distribution. As the sample number increasing, the student's T distribution is closer to Gussian distribution. From analysis aspect we choose student T distribution to analyse, in code practice, we directly using the confindence table value from Wiki, we have 95% confidence the $E(result)$ is in the interval of $trulyvalue \pm 2\sigma$. Assume our sample follow the Gussian distribution $N(\mu, \sigma^2)$. Consider the current senario, we can get the current σ from current code context online, and we just calculate the interval of μ is including pi or other hypersphere volume by calculate. For example, if we use 2 dimension, the confidence interval is $Pi \pm digit_4(Pi)$. ($digit_4(x)$ means get the 4 digits percision error of x, like the return value of Pi is 0.0005). And because we also know our sample follows Gassian distribution, we can go the following confidence interval step.

Distribution Derivation

The senario is we know every sample(this sample means every estimate of current hypersphere volume) observed value, the sample number n, and we need to make sure the confidence interval less than the value of 4 digits percision:.

$$ConfidenceInterval \leq 2 \times digit_4(volume)$$

According to this condition, the best estimate of $\overline{X}$ is the student's t distribution according to pivot varialble T. The pivot variable T is as follows:

$$T = \frac{\sqrt{n}(\overline{X} - \mu)}{S}$$

T follows the student's t distribution t_{n-1}. Thus, the confidence interval about variable x is as follows:

$$[\overline{X} - \frac{S}{\sqrt{(n)}} t_{n-1}(\alpha/2), \overline{X} + \frac{S}{\sqrt{(n)}} t_{n-1}(\alpha/2)]$$

Thus, we could just make sure the terminal condition is confidence interval less than 4 digits percision.

$$2 \times \frac{S}{\sqrt{(n)}} t_{n-1}(\alpha/2) \leq 2 \times digits_4(HyperVolume)$$

For example, in our code we use 100 sample to test, in 2 dimension terminal condition is that confidence interval less than 2×0.0005. If the sample number beyond 20, we can consider student's t distribution is close enough to Gassian distribution.Thus, we can consider $t_{n-1}(\alpha/2)$ is same as $u_{0.025} = 1.96$. So once the confidence interval meet the terminal condition, break out and get the result.

Data Analysis

We can see that with the increase of d, the time complexity of the Cube based Integration method is much greater than that of the Monte Carlo Integration method.This is because for each increment of 1, the Monte Carlo Integration method simply adds a number to the length of the side but the Cube based Integration method adds an extra layer of recursion.

The data also shows that for the same d and accuracy, the Cube based Integration method requires more total sample count to achieve the same effect as the Monte Carlo Integration method.

If we only analyze the chart of the Monte Carlo Integration:

dimension	*samplesize*	*timecost*
1	4×10^3	$7ms$
2	4×10^7	$567ms$
3	4×10^7	$777ms$
4	4×10^7	$936ms$
5	4×10^8	$9s$
6	4×10^8	$10s$
7	4×10^9	$102s$
8	4×10^9	$104s$
9	4×10^9	$104s$
10	4×10^{10}	$9972s$
11	4×10^{10}	$1034s$
12	4×10^{10}	$1065s$
13	4×10^{11}	$11319s$
14	4×10^{11}	$13432s$

Table 1: Monte Carlo Integration Method Time Cost.

1. In order to measure accurately, basically the sample count should be expanded by 10 times after every 2, 3 numbers.

2. With the increase of d, the time complexity is related to the size of the sample set but has little relation with d. For example, when $d = 7$ and $d = 8$, the datasets run by the program are both in size of 4×10^9. For each sample, the latter adds one more dimension than the former. However, they eventually take almost the same amount of time. So it can be seen that the size of each sample has little impact on the time complexity, but the sample count is the main factor.

Comparing with the Cube-base method, we can push Monte Carlo into much more higher dimension even to 17 dimension. On the other hand, Cube-base basic method can only keep 4 digits percision at 2 dimension. Even with the performance improvement method as we only consider the surface condition. The Cube-base method can only push into 3 dimension and cannot keep 4 digits percision. The result is as follows:

The improved cube-base method can only estimate to 3 dimension. Thus, we improve the cube-base method

D	$estimatesize$	$actuallysize$	$estimate$	$standard$	$time$
2	4^{13}	3.27×10^4	1	1	$10ms$

Table 2: Basic Cube-Base Time Cost.

D	$estimatenum$	$actualnum$	$estimate$	$standard$	$timecost$
2	4^{13}	3.27×10^4	3.1406	3.1415	$10ms$
3	8^{12}	2.64×10^7	4.1841	4.1887	$63s$

Table 3: Improved Cube-Base Time Cost.

further to generate the Modified Method. We can use this modified cube-base method to replace other hypercube volume estimate. The result is as follows:

$dimension$	$sample$	$estimate$	$standard$	$time$	$threads$
2	$1.63 \times 10^4 (2^{14})$	3.1417	3.1415	$6ms$	32
4	$3.27 \times 10^4 (2^{15})$	4.9351	4.9348	$150ms$	32
8	$6.55 \times 10^4 (2^{16})$	4.0591	4.0587	$751ms$	32
16	$1.31 \times 10^5 (2^{17})$	0.23536	0.23533	$4s$	32
32	$5.24 \times 10^5 (2^{19})$	4.3035×10^{-6}	4.3030×10^{-6}	$158s$	32
64	$2.09 \times 10^6 (2^{21})$	3.0807×10^{-20}	3.0805×10^{-20}	$6643s$	32

Table 4: Modified Method Time Cost.

As for the 8 digit precision, we find that the time complexity of the Monte Carlo Integration method becomes much higher when $d = 3$, and it cannot be tested higher than dimension 3. The same as Cube-base, but we can get the 7 digits percision result at 2 dimension, 8 digits need cost more time.

Performance Improvement Method:

Multi-threads

In Part 1 and Part 2, using the multi-threaded Monte Carlo Integration method, the number of samples is divided into 16 equal parts. Each thread is responsible for a part of the sample, and all the results are stored in the corresponding position of the array. After all the threads are finished, add the sum of the arrays as the final statistical result.

Pruning

If the sum is already greater than 1, it means that the point is definitely outside the hypersphere, and the remaining dimensions are not continued to be calculated.

In general, the monte carlo method time complexity is sames like not increasing as fast as Cube-Based method by dimension increasing.

Part 2

Monte Carlo Method

Basically, it is the same as Part 1. We removed the tests to find the right Nd and verify the 4-digit accuracy part.

dimension	*sample*	*estimate*	*time*
1	64	2.0	$0ms$
2	$16,777,216$	3.1425535678863525	$105ms$
3	$morethan64G$	$--$	$--$

Table 5: Monte Carlo for 8 digit.

dimension	*estimatesample*	*actuallysample*	*estimate*	*standard*	*timecost*
2	4^{24}	6.71×10^7	3.1415921	3.1415926	$85s$

Table 6: Improved Cube-Base Time Cost in 8 digits percision.

Cube-based Method

First, we calculate the dth root of 1000000, and then round off to an integer as K, the number of segments to cut the hypercube.

Since K must be an integer, the total number of small hypercubes, that is K^d, will be different from $n = 1000000$. The difference, we named sample count error, as follows:

$$sampleCount = n^{\frac{1}{d}}$$
$$sampleError = |n - sampleCount^d|$$

Data Analysis

For this part, we present the data in five terms, the relative error of the Monte Carlo Integration method, the relative error of the Cube based Integration method, their absolute difference, their relative difference, and Cube based Integration method's sample error.

We can find that this data can be divided into three stages.

In the first stage, when $d < 7$, the dimension of d is small. Both the Monte Carlo Integration method and the Cube based Integration method can work well. And Monte Carlo Integration method is more accurate than Cube based Integration method.

The second stage is that when $7 \leq d < 17$, the Monte Carlo Integration method is still more accurate at this stage, but the relative error of the Cube based Integration method exceeds 1 and becomes larger. There are two reasons:

1. The dth root of 100000 is rounded to a smaller number than itself so that the total sample size is less than 1,000,000; 2. Even if K is often rounded to a number of $K^d >> 1000000$, but we also verified in part1, the high dimensional Cube based Integration method requires a very large sample size. Even though this K^d is much larger than 1000000, it is still not enough to accurately estimate the volume of the hypersphere.

The third stage is $17 \leq d < 34$. In this stage, the Monte Carlo Integration method fails. We can observe the number of points that fall within the hypersphere:

This is because the point that falls into the hypersphere is almost zero. At the same time, the Cube based Integration method is seriously inaccurate.

The forth stage is when d>34, the Cube based Integration method has also completely failed at this stage, because

$dimension$	$numberofpointsfellintothehypersphere$
15	11
16	2
17	0
18	1

Table 7: The number of points fell into the hypersphere.

the 34th root and higher root of 1000000 are close to 1 or less. The sample size is $1^d = 1$. It is unable to estimate the volume. Therefore, we do not need to test higher-dimensional data, because for $N = 1000000$, both methods fail.

The above analysis conclusions can be simply expressed as the following table:

$dimension$	$MonteCarlo$	$Cube - Based$
$d < 7$	$workswell$	$workswell$
$7 \leq d < 17$	$workswell$	$inaccurate$
$17 \leq d < 34$	$fails$	$inaccurate$
$34 < d$	$fails$	$fails$

Table 8: The number of points fell into the hypersphere.

Conclusions of Part 1 and Part 2

If the accuracy and the dimension are the same, the Monte Carlo Integration method requires less samples and less time complexity than the Cube based Integration method. However, the advantage of the Cube based Integration method is that it is more stable than the Monte Carlo Integration method. It is unlike the Monte Carlo Integration method, which has a great deal of randomness and contingency.

Part 3

Based on the analysis of the first two parts, we can see that it is obviously the Monte Carlo Integration method. Because with the same N value, the Monte Carlo Integration method has a low time complexity and high precision.

If we can get at most N sample from outside, we also need to choose the dimension level as low as possible to estimate the accurate value of PI.

If we uniform the test standard, like only give 2^10 sample size to calculate π as accurate as possible. There is the result of Monte Carlo Method and Our Modified Cube-Based Method result:

If we want to implement the code of this part, we can easily get the π value from the above method $Answer.answer(intdimension)$ (The code is as $Answer.pi(intd, doublevolume)$). However, we can directly use the 2 dimension value as the pi because the radius r is 1.

For example, N=1000000, the data we measured were as followed:

As we can see, just as Part 2, when d approaches 17, the Monte Carlo Integration method fails because the point inside the hypersphere is close to zero.

Another method is using our modified method, it is only need 2^10 sample can get high percision of π estimate. The

d	$Result$	$error$	$thepointsinsidethehypercube$
2	3.140444	0.001149	785111
3	3.140046	0.001547	523341
4	3.1422896110957055	$6.969575059123656E-4$	308562
5	3.150942716077206	0.009350062487412991	165474
6	3.142104691963047	$5.120383732539757E-4$	80785
7	3.1444487766559206	0.0028561230661274806	37013
8	3.1343676527911053	0.0072250007986878195	15709
9	3.1311292889612523	0.010463364628540806	6357
10	3.1562413674065137	0.014648713816720615	2549
11	3.1095473972097993	0.03204525637999378	874
12	3.1220295936375675	0.019563059952225625	314
13	3.2204971689036377	0.07890451531384457	129
14	3.0821743912981296	0.05941826229166347	32
15	3.074143621509071	0.06744903208072195	10
16	3.1842529195438942	0.042660265954101106	4
17	3.1131030644786692	0.02848958911112387	1
18	0.0	0.0	0

Table 9: calculate pi by Monte Carlo Integration Method

result is as follows:

$dimension$	$sample$	$estimate$	$standard$
2	1024	3.1438	3.1415

Table 10: Pi estimate by Modified method.