Part 1:

1. Design and implementation

1.1 Monte Carlo Integration method

1.1.1 Basic ideas

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. Its code is:

for (int j = 0; j < sampleCount; j++) {

double sum = 0;

for (int i = 0; i < d && sum <= 1; i++) {

double randomValue = r.nextDouble();

sum += randomValue \* randomValue;

if (sum > 1) {

break;

}

}

if (sum <= 1) {

count++;

}

}

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.

1.1.2 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.

1.2 Cube based Integration method

Since it requires 4-bit precision, its minimum number of samples is at least =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. Its code is:

double unit = (double)1.0 / sampleCount;

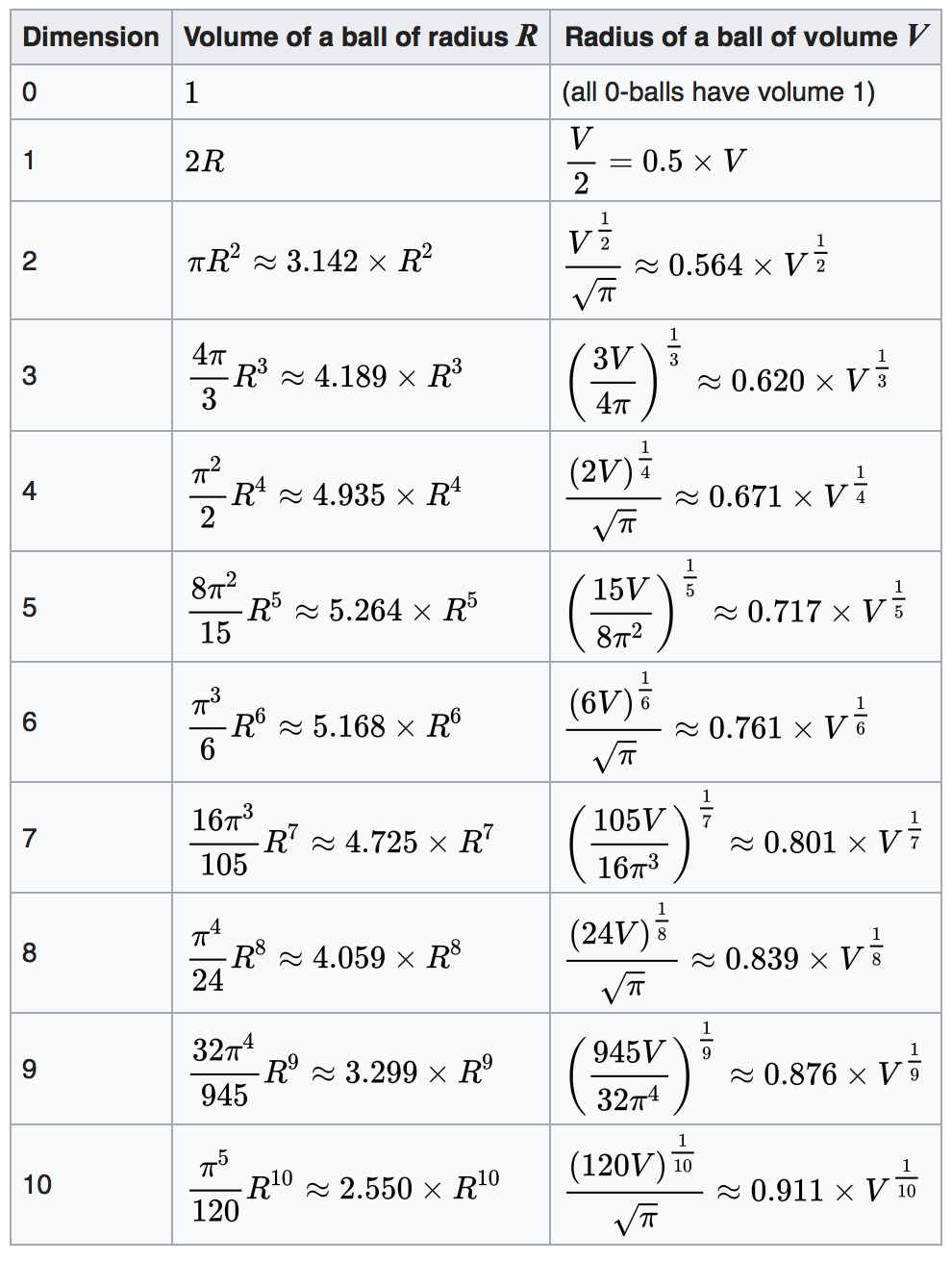
for (int j = 0; j < sampleCount; j++) {

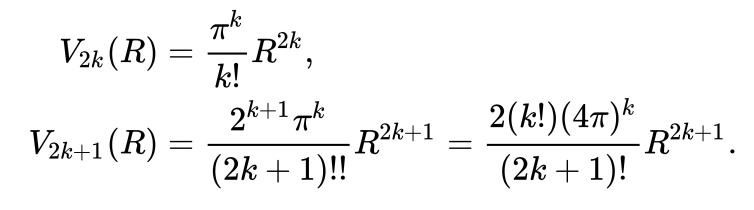
double temp = j \* unit;

helper(sum + temp \* temp, remain - 1, count, sampleCount);

}

1.3 answer() function：

I look at the table on Wikipedia (<https://en.wikipedia.org/wiki/Volume_of_an_n-ball>), and get the formula for the volume of an n-hypersphere:

I used this formula to write the answer() function as a checksum.

1.4 Time statistics:

When it starts the calculation, call System.currentTimeMillis() to record the start time. When the calculation is completed, it notes the current time again as the end time. The difference between the two times is the time used. Its code is:

start = System.currentTimeMillis();

… …

end = System.currentTimeMillis();

To be scientifically accurate, each d needs to test the same number of datasets more than once, and we take the time of the first dataset of that number as the time.

2.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 method:

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

| 表格 1-1 | | |
| --- | --- | --- |
| d | sample size(4\*10^) | time(ms) |
| **1** | 3 | 7 |
| **2** | 7 | 567 |
| **3** | 7 | 777 |
| **4** | 7 | 936 |
| **5** | 8 | 9593 |
| **6** | 8 | 10154 |
| **7** | 9 | 102634 |
| **8** | 9 | 104336 |
| **9** | 9 | 104972 |
| **10** | 10 | 997206 |
| **11** | 10 | 1034078 |
| **12** | 10 | 1065981 |
| **13** | 11 | 11219764 |
| **14** | 11 | 13432158 |

b)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.

c)As for the 8-digit precision, we find that the time complexity of the Monte Carlo Integration method becomes much higher when d=3.

3. Efficiency Improvement and optimization methods:

3.1 Multi-threaded

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.

3.2 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.

PART 2:

1. Design and implementation

1.1 Monte Carlo Integration 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.

1.2 Cube based Integration 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 1000000. The difference, we named sample count error. Its code is:

int sampleCount = (int)Math.round(Math.pow(1000000, (double)1.0 / (double)d));

int sampleCountError=(int)Math.abs(1000000-Math.pow(sampleCount,d));

2.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<=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<=d<34. In this stage, the Monte Carlo Integration method fails. We can observe the number of points that fall within the hypersphere:

|  |  |
| --- | --- |
| d | The number of points that fell into the hypersphere |
| 15 | 11 |
| 16 | 2 |
| 17 | 0 |
| 18 | 1 |

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 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:

|  |  |  |
| --- | --- | --- |
| d | Monte Carlo Integration method | Cube based Integration method |
| d<7 | works well | works well |
| 7<=d<17 | works well | inaccurate |
| 17<=d<34 | Fails | inaccurate |
| d>34 | Fails | Fails |

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.