**Project MC: Monte Carlo vs. Deterministic Volume Integration**

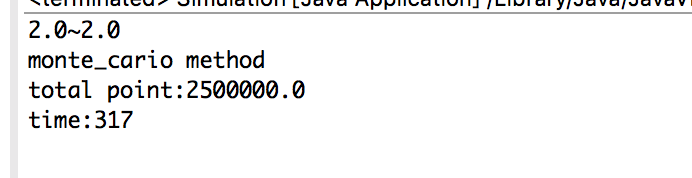
Name: Wenbo zhao ID: 000996017

**Introduction**

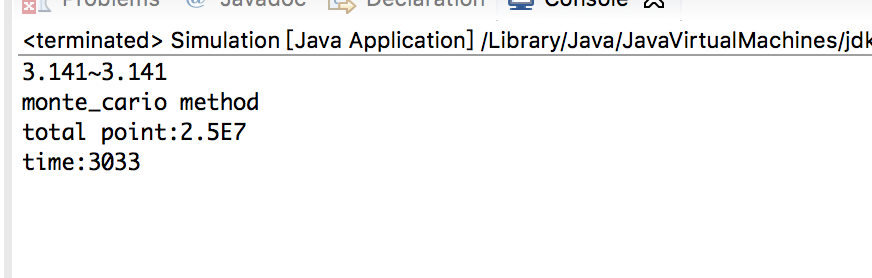
**Monte Carlo Integration:**

In Monte Carlo method, we need to make sure that the result is correct to 4 digits with 99% confidence. Thus, I wrote a program which can first run the samples for 25 rounds, and If the result doesn’t reach standard, it then automatically runs 25+10 rounds. And If again not valid, it runs 25+10+10 rounds. It runs until the result is correct to 4 digits with 99% confidence.

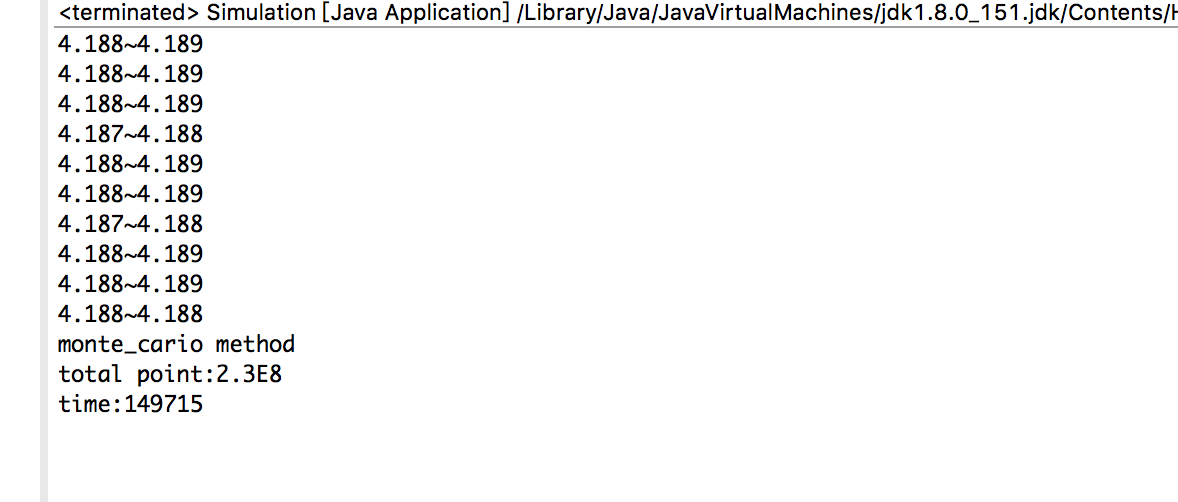
D=1



D=2



D=3



D=4



D=5



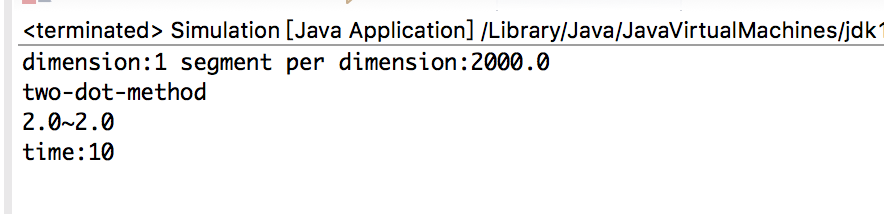
**Cube-based Integration:**

In the Cube-based Integration, I found two methods to judge the relative location of the “small” hypercube and the hypersphere. One is Two-dot-exam method, which can classify the hypercube accurately but has a low speed. The other is Mixed-method, which classify the “inside” hypercube with small deviation but runs much faster than Two-dot-exam method.

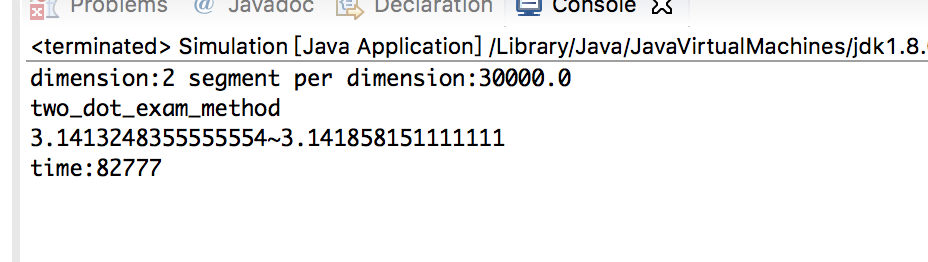
**Note:** Both of the Two-dot-exam method and Mixed-method can be times faster than the original method if using symmetry of a hypersphere. However, the assignment asks us to view the classification as a black box, which means we can’t use the symmetry of a hypersphere.

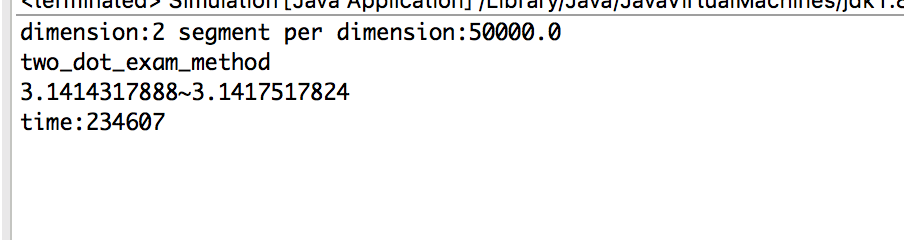
* **Two-dot-exam method:**

D=1



D=2



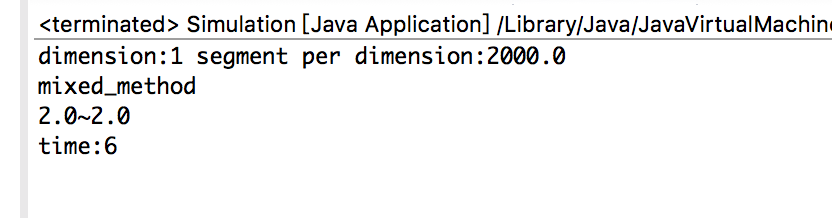


D=3

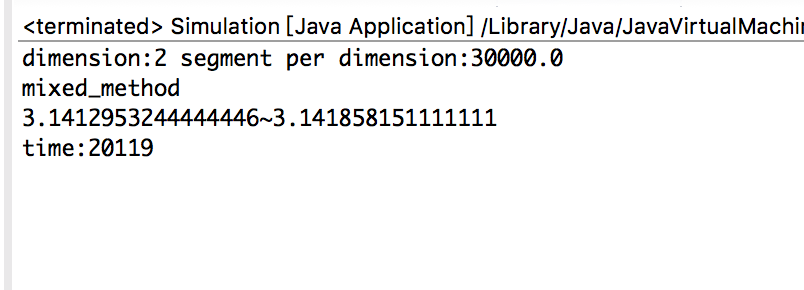
too long time to reach 4 digits accuracy not using symmetry, so we use Mixed-method to calculate three dimensions.

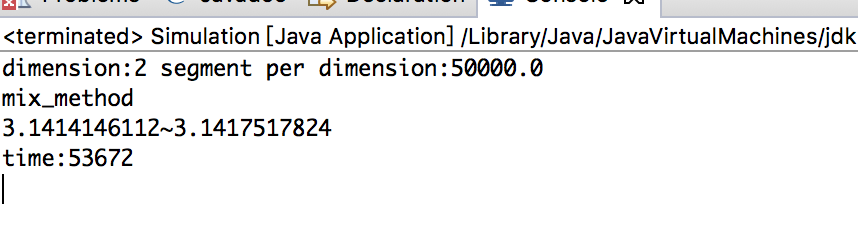
* **Mixed-method:**

D=1

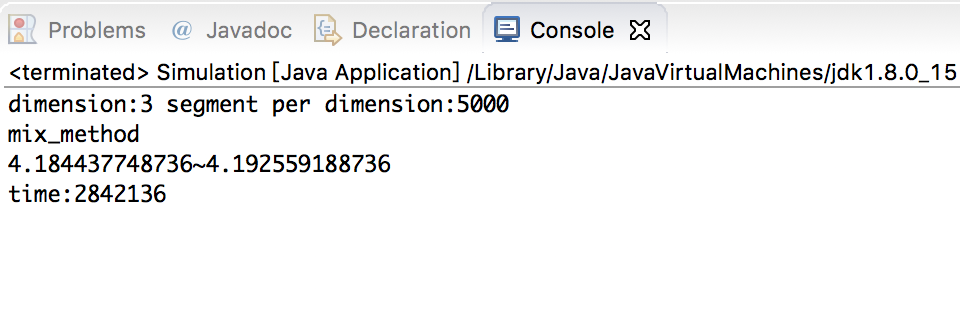


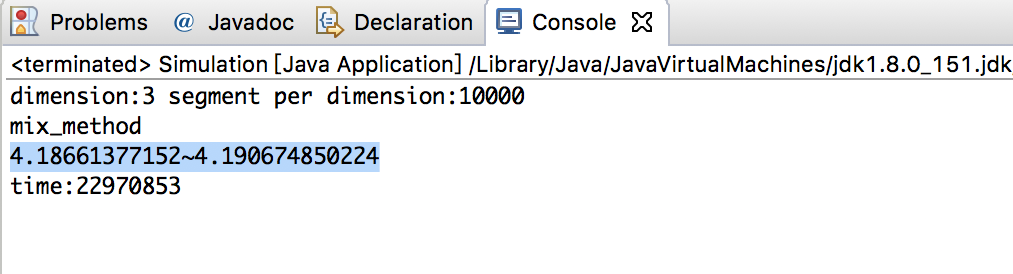
D=2



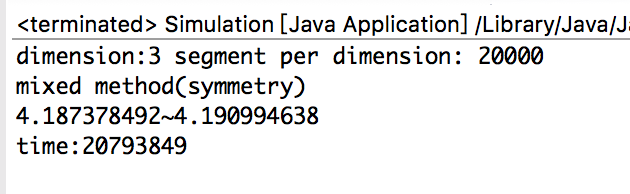


D=3





For the time is still too long, I use symmetry of a hypersphere and let k=20000, the result will be:

****

**Part 1**

The Monte Carlo Integration can push d to 5, even 6 using my MacBook Pro allocating 9G memory. The Cube-based Integration, using Mixed-method, can only nearly push d to 3. Using Two-dot-exam method, is even worse. (But if you use symmetry of hypersphere, you can definitely push d to 3). For running details, you can see running results in Introduction.

As 4 digits accuracy still is too wide, I decided to give a figure which can show the relationship between Accuracy, Dimensions and Run time.

**Monte Carlo method:**

**Note:** Accuracy means the difference between two boundaries of confidence Interval.

**Cube-based method:**

To run an accurate result, we use Two-dot-exam method here, but considering the running speed, I make a trade-off to use symmetry of hypersphere.

Keep the accuracy the same value in both methods, we can get an approximately figure of the Dimensions and Run time. The shape is like this:

As for 8 digits of precision with 99.99% confidence, Monte Carlo also has a smooth curve. Cube-based method, however, will has a much steeper curve. (sorry for not have so much time to run enough data).

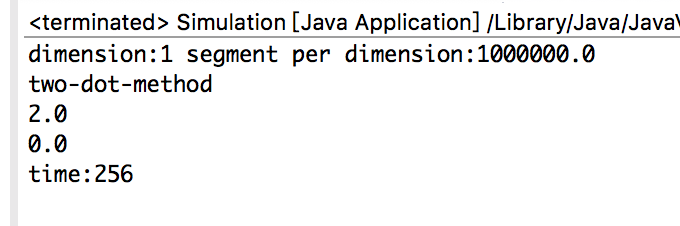
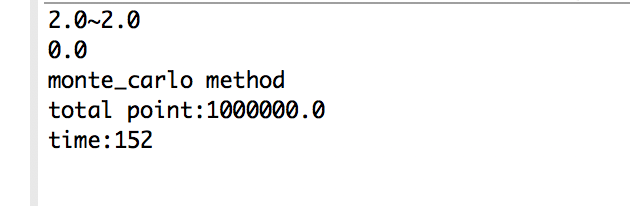
**Part 2**

**Monte Carlo Integration V.S Cube-based Integration (Two-dot-exam method using symmetry of hypersphere)**

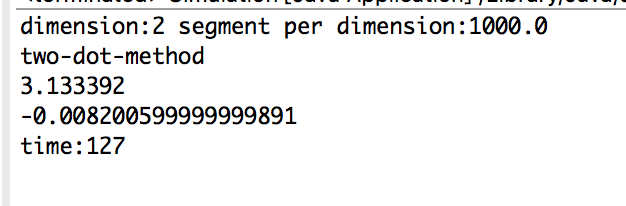
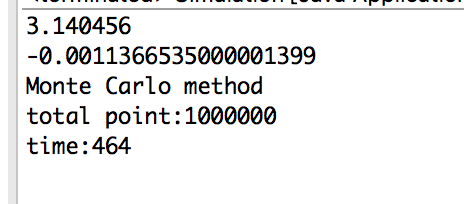
The accuracy in this part seems to have different meaning in Part 1. This accuracy here means the difference between the estimate result and the real one. In Monte Carlo method, I set N=1000000 and runs only one round. In Cube-based method, we estimate the volume of the hypersphere as the volume of small hypercubes inside the hypersphere plus half of those that are bisected by its boundary.

In part one, reaching 4 digits requires both the upper and lower boundary reach 4 digits. In part two, however, the estimate result is between the two boundaries. More specifically, the estimate result surrounds the mean of two boundaries. Thus, the method in part one is more accurate.

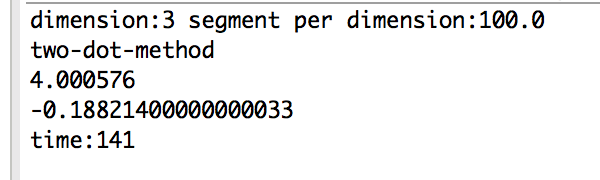
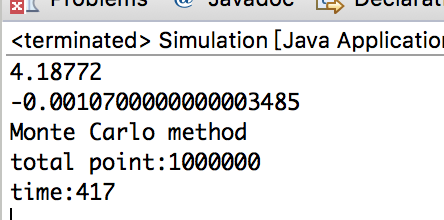
D=1, N=1000000, k=1000000

****

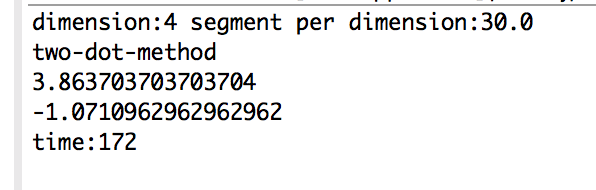
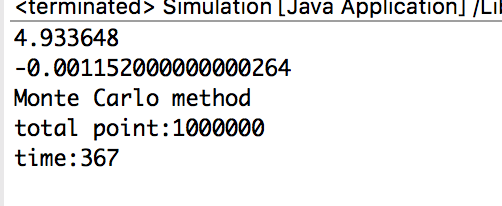
D=2, N=1000000, k=1000



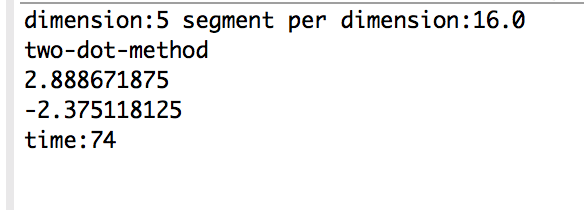
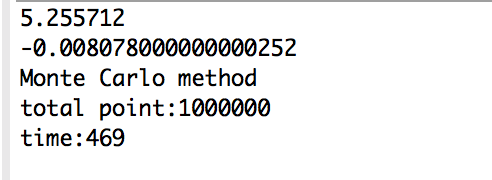
D=3, N=1000000, k=100



D=4, N=1000000, k=30



D=5, N=1000000, k=16



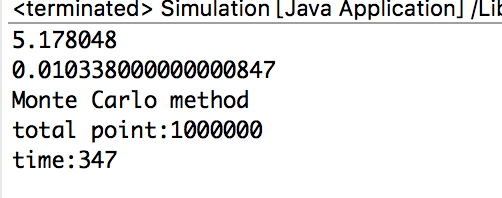
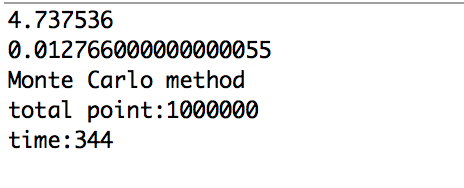
Conclusion: We can see from the figures that Monte Carlo gives a more accurate answer and has less time to run.

**Part 3**

Obviously, through previous analysis, Monte Carlo method is more accurate and should be used to compute .

Additional data for Monte Carlo method:

D=6, N=1000000 D=7, N=1000000

****

To exam which dimension can compute more accurate, we can use the Monte Carlo results in Part 2(from 2 dimensions to 5 dimensions).

Accuracy of equals:

2 dimensions: Accuracy

3 dimensions: Accuracy/ mean = Accuracy /4.188792

4 dimensions: Accuracy/ mean = Accuracy /4.9348

5 dimensions: Accuracy/ mean = Accuracy /5.263789

6 dimensions: Accuracy/ mean = Accuracy /5.1677

7 dimensions: Accuracy/ mean = Accuracy /4.72477

Through previous analysis, Monte Carlo has very close Accuracy. Therefore, if N is pretty large, we can assume all Accuracy in 2-7 dimensions are the same. Then the bigger the denominator is, the smaller the Accuracy is (also means more accurate).

In conclusion, choosing 5 will get a more accurate result for estimating .

At last, about variance reduction techniques, we can use the Antithetic variables because we can run the system 2 times. One use uniform random number, the other use (1-unifrom random number).

**Appendix (code)**

**Cube\_based\_Integration.java:**

import java.util.ArrayList;

import java.util.Random;

public class Cube\_based\_Integration {

ArrayList<Double> SCube\_Center\_Coordinate;

int dimension;

double segment;

double hypercube\_volume;

double in\_hypersphere\_Scube = 0;

// double cross\_hypersphere\_Scube = 0;

double pass\_hypersphere\_Scube = 0;

double out\_hypersphere\_Scube = 0;

double total\_hypersphere\_Scube = 0;

double UpperBounder = 0.0;

double LowerBounder = 0.0;

public double getUpperBounder() {

return UpperBounder;

}

public double getLowerBounder() {

return LowerBounder;

}

public Cube\_based\_Integration(int dimension, double segment) {

this.dimension = dimension;

this.segment = segment;

hypercube\_volume = Math.pow(2, dimension);

total\_hypersphere\_Scube = Math.pow(segment, dimension)/Math.pow(2, dimension);

SCube\_Center\_Coordinate = new ArrayList<Double>();

for (int i = 0; i <= dimension; i++) {

SCube\_Center\_Coordinate.add(1.0 / segment);

}

}

public void CalculateVolume() {

// set a redundant bit, which is used to check if has finished loop

int lastindex = SCube\_Center\_Coordinate.size() - 1;

judgeSmallCubeLocation();

//calculate\_in\_out\_cube();

do {

double temp = SCube\_Center\_Coordinate.get(0) + 2.0 / segment;

SCube\_Center\_Coordinate.set(0, temp);

for (int i = 0; i < dimension; i++) {

if (SCube\_Center\_Coordinate.get(i) >= 1.0) {

SCube\_Center\_Coordinate.set(i, 1.0 / segment);

temp = SCube\_Center\_Coordinate.get(i + 1) + 2.0 / segment;

SCube\_Center\_Coordinate.set(i + 1, temp);

}

}

if (SCube\_Center\_Coordinate.get(lastindex) < 3.0 / segment) {

judgeSmallCubeLocation();

//calculate\_in\_out\_cube();

}

} while (SCube\_Center\_Coordinate.get(lastindex) < 3.0 / segment);

UpperBounder = (total\_hypersphere\_Scube - out\_hypersphere\_Scube) / total\_hypersphere\_Scube \* hypercube\_volume;

LowerBounder = in\_hypersphere\_Scube / total\_hypersphere\_Scube \* hypercube\_volume;

System.out.println("dimension:" + dimension + " segment per dimension:" + segment);

System.out.println("two-dot-method");

double temp= (in\_hypersphere\_Scube +1/2\*pass\_hypersphere\_Scube) / total\_hypersphere\_Scube \* hypercube\_volume;

System.out.println(temp);

System.out.println(temp-4.18879);

// System.out.println(UpperBounder-LowerBounder);

// System.out.println(LowerBounder + "~" + UpperBounder);

}

public void calculate\_in\_out\_cube() {

double tempdouble = 0.0;

double tempdouble2 = 0.0;

double sum = 0.0;

double sum2 = 0.0;

double smallHypersphereR = 0.0;

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

tempdouble = Math.abs(SCube\_Center\_Coordinate.get(j) - 1) - (1.0 / segment);

tempdouble2 = Math.pow((SCube\_Center\_Coordinate.get(j) - 1), 2);

// System.out.println(tempdouble);

if (tempdouble < 0) {

tempdouble = 0;

}

sum = sum + Math.pow(tempdouble, 2);

sum2 = sum2 + tempdouble2;

}

if (sum > 1.0 \* 1.0) {

out\_hypersphere\_Scube++;

}

sum2 = Math.sqrt(sum2);

smallHypersphereR = Math.sqrt(dimension) / segment;

if (sum2 + smallHypersphereR <= 1) {

in\_hypersphere\_Scube++;

}

}

public void judgeSmallCubeLocation() {

ArrayList<Double> tempCoordinate = new ArrayList<Double>();

ArrayList<Double> tempCoordinate2 = new ArrayList<Double>();

double sum = 0.0;

double sum2 = 0.0;

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

if ((SCube\_Center\_Coordinate.get(j) - 1) <= 0) {

tempCoordinate.add(SCube\_Center\_Coordinate.get(j) - (1.0 / segment));

tempCoordinate2.add(SCube\_Center\_Coordinate.get(j) + (1.0 / segment));

} else {

tempCoordinate.add(SCube\_Center\_Coordinate.get(j) + (1.0 / segment));

tempCoordinate2.add(SCube\_Center\_Coordinate.get(j) - (1.0 / segment));

}

sum = sum + Math.pow(tempCoordinate.get(j) - 1, 2);

sum2 = sum2 + Math.pow(tempCoordinate2.get(j) - 1, 2);

}

if (sum > 0) {

if (sum <= 1 \* 1) {

in\_hypersphere\_Scube++;

} else if (sum > 1 \* 1) {

if (sum2 < 1 \* 1) {

pass\_hypersphere\_Scube++;

} else if (sum2 >= 1 \* 1) {

out\_hypersphere\_Scube++;

}

}

}

}

}

**Monte Carlo Integraion.java:**

import java.util.ArrayList;

public class Monte\_Carlo\_integration {

ArrayList<Double> Coordinate;

int dimension;

double testing\_num;

double hypercube\_volume;

// Random r;

public Monte\_Carlo\_integration(int dimension, double testing\_num) {

this.dimension = dimension;

this.testing\_num = testing\_num;

hypercube\_volume = Math.pow(2, dimension);

}

public double CalculateVolume() {

double point\_in\_hypersphere = 0;

for (int i = 0; i < testing\_num; i++) {

// GenCoordinate();

Coordinate = new ArrayList<Double>();

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

double tempDouble = Math.random() \* 2.0;

// System.out.println("tempdouble"+tempDouble);

Coordinate.add(tempDouble);

}

double sum = 0.0;

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

// System.out.println(Coordinate.get(j));

sum = sum + Math.pow(Coordinate.get(j) - 1, 2);

}

if (sum <= 1 \* 1) {

point\_in\_hypersphere++;

}

}

double volume = point\_in\_hypersphere / testing\_num \* hypercube\_volume;

return volume;

}

public void multiruns() {

int round = 15;

double lower = 0.0;

double higher = 0.0;

// do {

round = round + 10;

double mean = 0.0;

double variance = 0.0;

double sum = 0.0;

double squaresum = 0.0;

double temp = 0.0;

double accuracy =0;

ArrayList<Double> volume = new ArrayList<Double>();

for (int i = 0; i < round; i++) {

temp = CalculateVolume();

volume.add(temp);

sum = sum + temp;

}

mean = sum / round;

for (int i = 0; i < round; i++) {

temp = volume.get(i);

squaresum = squaresum + Math.pow((temp - mean), 2);

}

variance = squaresum / (round - 1);

lower = mean - Math.sqrt(variance / round);

higher = mean + Math.sqrt(variance / round);

System.out.println(lower + "~" + higher);

System.out.println(higher-lower);

lower = Math.floor(lower \* 10000) / 10000;

higher = Math.floor(higher \* 10000) / 10000;

//System.out.println(lower + "~" + higher);

// if (lower == higher) {

// System.out.println("is correct to 4 digits with 99% confidence? :" + true);

// } else {

// System.out.println("is correct to 4 digits with 99% confidence? :" + false);

// }

// } while (lower != higher || lower != 3.1415);

System.out.println("monte\_carlo method");

System.out.println("total point:" + testing\_num \* round);

}

}

**Simulation.java:**

import java.util.ArrayList;

public class Simulation {

static double volume;

public Simulation() {

}

public static void main(String[] args) {

long start = System.currentTimeMillis();

Monte\_Carlo\_integration mci = new Monte\_Carlo\_integration(5, 1000000);

double temp = mci.CalculateVolume();

System.out.println(temp);

System.out.println(temp-5.26379);

System.out.println("Monte Carlo method");

System.out.println("total point:" + 1000000);

// Cube\_based\_Integration cbi = new Cube\_based\_Integration(3,100);

// cbi.CalculateVolume();

System.out.println("time:" + (System.currentTimeMillis() - start));

}

}