レポート

計算物理演習レポート1 モンテカルロ法

1 概要

モンテカルロ法とは数値計算を乱数を用いて行う手法の総称 [1] である。今回はこのモンテカルロ法を用いて n 次元球の体積を導出する。

なお、各値の計算には C++ 14、プロットには Python 3 の matplotlib を用いた。また、乱数の計算には MT 法を用いた。

2 2 次元においてのプロット

2.1 内容

まずは 2 次元球、つまり円の場合において、$x : [0, 1)$, $y : [0, 1)$ にランダム点を打ったものをプロットした。

2.2 ソースコード

sphere2d.cpp

```
#include <iostream>
//#include <cstdlib>
#include <cmath>
#include <random>
#include <fstream>
#include <string>
#include "util.h"

using namespace std;

int main() {
    int n = int(pow(10, 5)), d = 2, count = 0, interval = pow(10, 4);
    double r2s[n];
    string s;
    for (int i = 0; i < n; i++) {
        double r2 = 0.0;
        for (int j = 0; j < d; j++) {
            double x_j = get_random_0_to_1();
            r2 += pow(x_j, 2);
            if (j == 0) {
                s += to_string(x_j) + ",";
            } else {
                s += to_string(x_j) + "\n";
            }
        }
        if (r2 <= 1) count++;
        if (i % interval == interval - 1 or i == n - 1) {
            r2s[i] = r2;
            double v = double(count) / double(i + 1) * pow(2, d);
            double p = double(count) / double(i + 1);
            double q = 1.0 - p;
            double error = 1.96 * pow(2, d) * sqrt(p * q / double(i + 1));
            cout << "i + 1 = " << i + 1 << " v = " << v << " ± " << error << "\n";
        }
    }
    write_to_file(s, "../out/x_y_plot_2d.csv");
}
```

util.cpp

```
//#include <random>
#include "Sample.h"

using namespace std;
```

util.h

```
#pragma once

#include <fstream>
#include <random>

using namespace std;

inline double get_random_0_to_1() {
```

```
  // 乱数生成器
  static mt19937_64 mt64(0);

  // [0.0, 1.0) の一様分布実数生成器
  uniform_real_distribution<double>
      get_rand_uni_real(0.0, 1.0);
  // 乱数を生成
  return get_rand_uni_real(mt64);
}

inline int write_to_file(const string& s,
    const string& fpath) {
  ofstream f;
  f.open(fpath);
  f << s;
  f.close();
  return 0;
}
```

plot2d.py

```
# -*- coding: utf-8 -*-

import matplotlib.pyplot as plt
import numpy as np
import csv

def main():
  data = []
  with open("../../cpp/out/x_y_plot_2d.
      csv") as f:
      rows = csv.reader(f, quoting=csv.
          QUOTE_NONNUMERIC)
      for row in rows:
          data.append(row)

  # print(data)
  x = [row[0] for row in data]
  y = [row[1] for row in data]
  plt.plot(x, y, 'b+', markersize=0.1)

  theta = np.linspace(0, np.pi / 2,
      100)
  r = np.sqrt(1.0)
  circle_x = r * np.cos(theta)
  circle_y = r * np.sin(theta)
  plt.plot(circle_x, circle_y, 'black')

  plt.axes().set_aspect('equal', '
      datalim')
  plt.grid(True, which="both", axis='
      both', ls="--", color="g")
  plt.xlabel('x')
  plt.ylabel('y')
  plt.savefig('out/plot2d.pdf')
  plt.savefig('out/plot2d.png')
  plt.show()


if __name__ == "__main__":
  main()
```

2.3 結果

C++ コンソール

```
i + 1 = 10000 v = 3.1444 ± 0.0321485
i + 1 = 20000 v = 3.142 ± 0.0227556
i + 1 = 30000 v = 3.14307 ± 0.0185714
i + 1 = 40000 v = 3.1426 ± 0.0160865
i + 1 = 50000 v = 3.13664 ± 0.0144245
i + 1 = 60000 v = 3.13553 ± 0.0131738
i + 1 = 70000 v = 3.13543 ± 0.0121971
i + 1 = 80000 v = 3.13605 ± 0.0114064
i + 1 = 90000 v = 3.13809 ± 0.0107448
i + 1 = 100000 v = 3.13872 ±
    0.0101907

Process finished with exit code 0
```

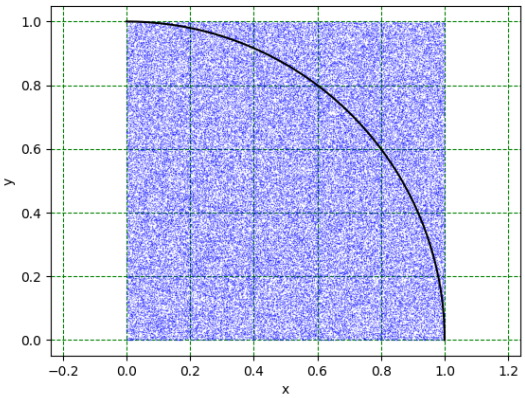

図 1　2 次元においてのランダム点プロット

2.4 考察

プロットされた点は偏りがなく、均等でなく見えるため、ランダムに点が打たれているできているといえる。乱数の評価という点において、MT 法によって生成された（疑似）乱数がモンテカルロ法を使うにあたって妥当であることが目視的に確認できる。

3 分散の収束

3.1 内容

分散が次第に小さくなっていき、超球の体積が理論値へ収束してくことを視覚的に確認する。次元は 2 次元から 20 次元までとし、試行回数は 1 億回、x 軸のみ対数グラフとしてプロットした。

求める超球の体積は次元を d とすると、

$$V_n(R) = 2^d \frac{(\text{球の中に入った点の個数})}{(\text{試行回数})} \tag{1}$$

と表せる。

求めた超球の体積の誤差は、p =(球の中に入った点の個数)/試行回数、n = 試行回数として、

$$1.96\sigma = 2^d \sqrt{\frac{p\,(1-p)}{n}} \tag{2}$$

と 95%信頼区間で求めた。

理論値は超球の体積の公式 [2]

$$V_n(R) = \frac{\pi^{n/2}}{\Gamma\left(\frac{n}{2}+1\right)} R^n \tag{3}$$

を用いた

3.2 ソースコード

sphere_multi.cpp

```
//#include <iostream>
#include <cmath>
#include <tuple>
#include "Sample.h"

using namespace std;

inline tuple<double, double, double, double> calc(int d, int i, int count) {
  double v = double(count) / double(i + 1) * pow(2, d);
  double p = double(count) / double(i + 1);
  double q = 1.0 - p;
  double error = 1.96 * pow(2, d) * sqrt(p * q / double(i + 1));
  return forward_as_tuple(v, p, q, error);
}

inline int main_inline() {
  const int n_pow = 8, n = int(pow(10, n_pow)), num_plot = 100, interval = n / num_plot;
  const int d_max = 20;
  string v_s;

  for (int d = 2; d <= d_max; d++) {
      double v_theory = pow(M_PI, double(d) / 2.0) / tgamma(double(d) / 2.0 + 1.0);

      int i_log = 0, count = 0;
      double r2s[n];
      string s;

      for (int i = 0; i < n; i++) {
          Sample sample(d);
          if (sample.run()) count++;

          double log10_i = log10(i + 1);
          if (log10_i >= double(n_pow) / double(num_plot) * double(i_log) or i == n - 1) {
              r2s[i] = sample.r2;
              double v, p, q, error;
              tie(v, p, q, error) = calc(d, i, count);
              printf("d = %d, i + 1 = %d, v = %f ± %f\n", d, i + 1, v, error);
              s += to_string(i + 1) + "," + to_string(v) + "," + to_string(error) + "," + to_string(v_theory) + "\n";
              i_log++;
          }
          if (i + 1 == int(pow(10, 8))) {
              double v, error;
              tie(v, ignore, ignore, error) = calc(d, i, count);
              v_s += to_string(d) + "," + to_string(v) + "," + to_string(error) + "\n";
          }
      }
      string fpath = "../out/r2_plot_" + to_string(d) + "d.csv";
      write_to_file(s, fpath);
  }

  string v_s_path = "../out/v_s_plot.csv";
  write_to_file(v_s, v_s_path);

  return 0;
}

int main() {
  clock_t start = clock();

  main_inline();

  clock_t end = clock();
  const double time = double(end - start) / CLOCKS_PER_SEC;
  printf("Process finished. (%2.1f s)\n
```

```
        ", time);
65
66    return 0;
67  }
```

Sample.cpp

```
1  //#include <cmath>
2  #include "Sample.h"
3  //#include "util.h"
4
5  Sample::Sample(int d) {
6    this->d = d;
7    this->r2 = 0.0;
8  }
9
10 Sample::~Sample() = default;
```

Sample.h

```
1  #include <cmath>
2  #include "util.h"
3
4  class Sample {
5  private:
6    int d;
7
8  public:
9    explicit Sample(int d);
10   ~Sample();
11
12   bool run();
13
14   double r2;
15 };
16
17 inline bool Sample::run() {
18   for (int j = 0; j < d; j++) {
19       double x_j = get_random_0_to_1();
20       r2 += pow(x_j, 2);
21       if (r2 > 1) return false;
22   }
23   return true;
24 }
```

util.cpp、util.h は 2 と同一である。

plotmulti.py

```
1  # -*- coding: utf-8 -*-
2
3  import matplotlib.pyplot as plt
4  import numpy as np
5  import csv
6
7  from matplotlib import ticker
8
9
10 def loop(d: int):
11   data = []
12   with open("../../cpp/out/r2_plot_{}d.
         csv".format(d)) as f:
13       rows = csv.reader(f, quoting=csv.
             QUOTE_NONNUMERIC)
14       for row in rows:
15           data.append(row)
16
17   # print(data)
18   n = [row[0] for row in data]
19   v = [row[1] for row in data]
20   error = [row[2] for row in data]
21   v_theory = [row[3] for row in data]
22   v_plus_error = [e + error[i] for i, e
         in enumerate(v)]
23   v_minus_error = [e - error[i] for i, e
          in enumerate(v)]
24
25   plt.plot(n, v, 'b', linestyle='solid')
26   plt.plot(n, v_theory, 'r', linestyle='
         solid')
27   # plt.plot(n, v_plus_error, 'b',
         linestyle='solid')
28   # plt.plot(n, v_minus_error, 'b',
         linestyle='solid')
29   plt.fill_between(n, v_plus_error, v,
         facecolor='lightblue')
30   plt.fill_between(n, v, v_minus_error,
         facecolor='lightblue')
31
32   # plt.axes().set_ylim(ymin=-max(v
         )*0.1, ymax=max(v_plus_error)*1.1)
33   plt.axes().set_ylim(ymin=-0.1*v_theory
         [0], ymax=2.1*v_theory[0])
34   plt.xscale('log')
35   # plt.yscale('log')
36   plt.grid(True, which="both", axis='
         both', ls="--", color="g")
37   plt.xlabel('n')
38   # plt.ylabel('v')
39   plt.title('d = {}'.format(d))
40   plt.savefig('out/r2plot{}d.eps'.format
         (d))
41   plt.savefig('out/r2plot{}d.png'.format
         (d))
42   plt.show()
43
44
45 def main():
46   for d in range(2, 21):
47       loop(d)
48
49
50 if __name__ == "__main__":
51   main()
```

3.3 結果

C++ コンソール：(長文より冒頭のみ)

```
d = 2, i + 1 = 1, v = 0.000000 ±
    0.000000
d = 2, i + 1 = 2, v = 2.000000 ±
    2.771859
d = 2, i + 1 = 3, v = 2.666667 ±
    2.133778
d = 2, i + 1 = 4, v = 3.000000 ±
    1.697410
d = 2, i + 1 = 5, v = 2.400000 ±
    1.717658
d = 2, i + 1 = 6, v = 2.666667 ±
    1.508809
d = 2, i + 1 = 7, v = 2.857143 ±
    1.338656
d = 2, i + 1 = 8, v = 3.000000 ±
    1.200250
d = 2, i + 1 = 9, v = 2.666667 ±
    1.231937
d = 2, i + 1 = 10, v = 2.800000 ±
    1.136124
```

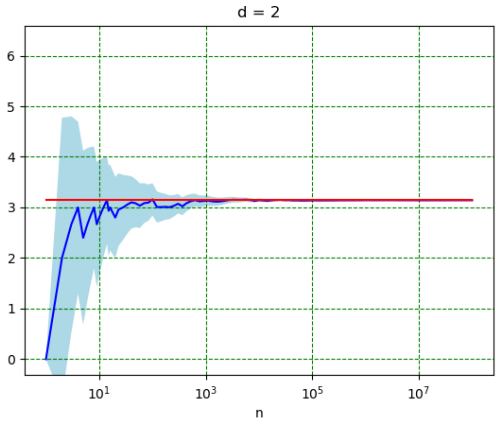

図 2　2 次元球の体積が理論値へと収束していく様子

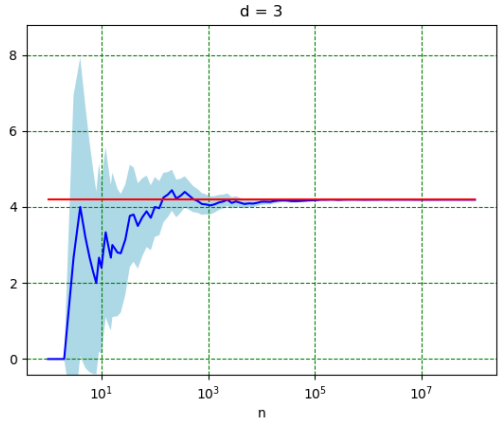

図 3　3 次元球の体積が理論値へと収束していく様子

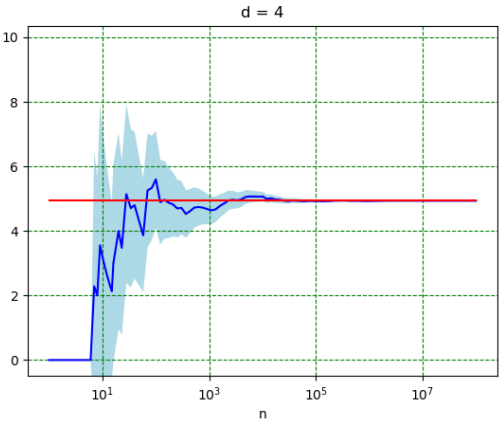

図 4　4 次元球の体積が理論値へと収束していく様子

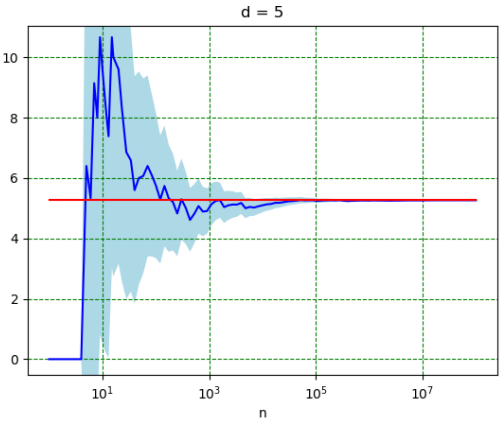

図 5　5 次元球の体積が理論値へと収束していく様子

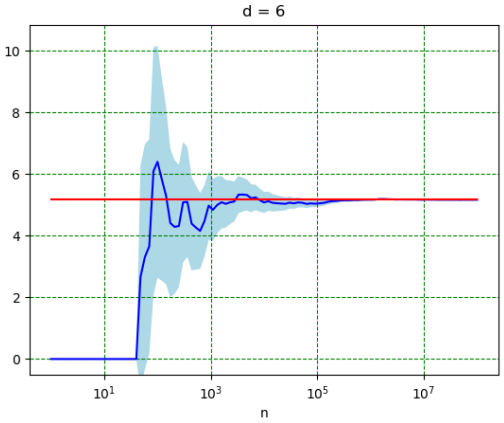

図 6　6 次元球の体積が理論値へと収束していく様子

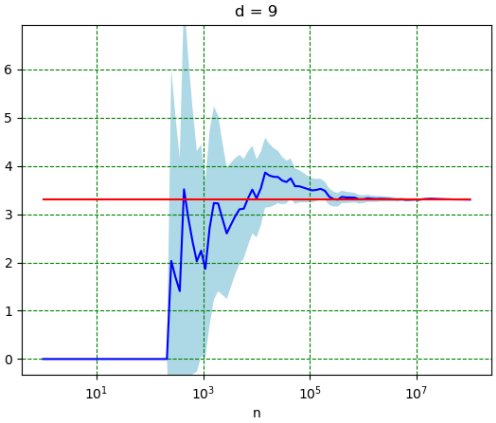

図 9　9 次元球の体積が理論値へと収束していく様子

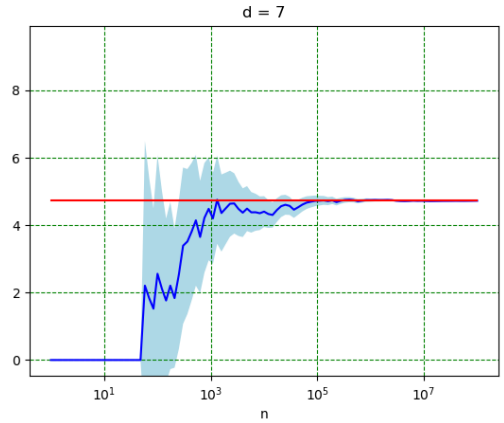

図 7　7 次元球の体積が理論値へと収束していく様子

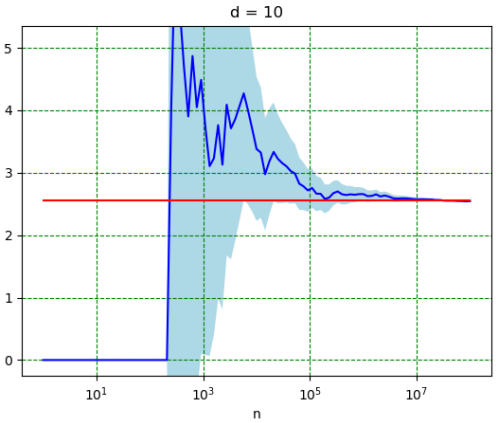

図 10　10 次元球の体積が理論値へと収束していく様子

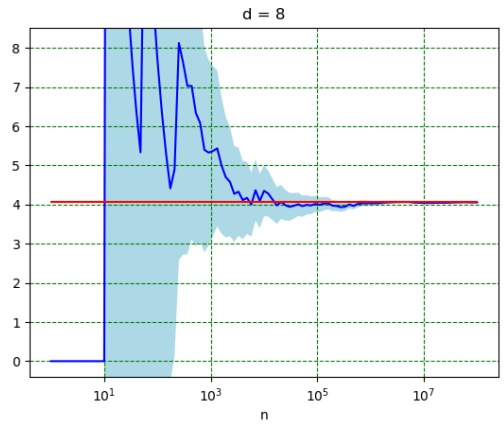

図 8　8 次元球の体積が理論値へと収束していく様子

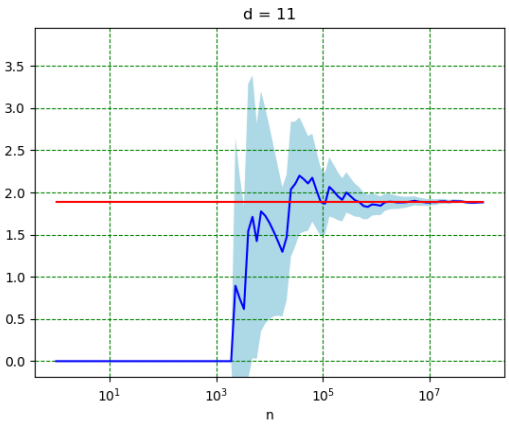

図 11　11 次元球の体積が理論値へと収束していく様子

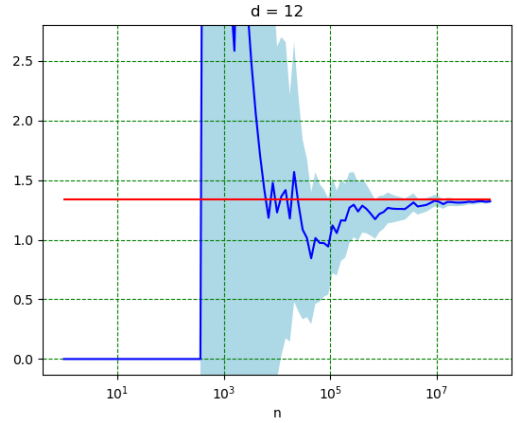

図 12　12 次元球の体積が理論値へと収束していく様子

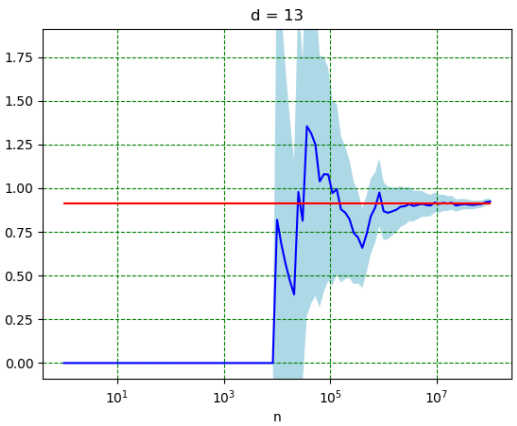

図 13　13 次元球の体積が理論値へと収束していく様子

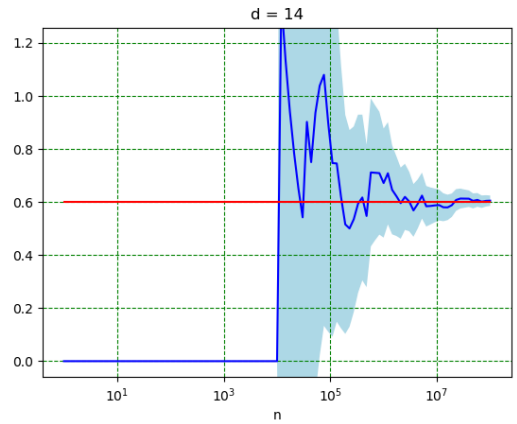

図 14　14 次元球の体積が理論値へと収束していく様子

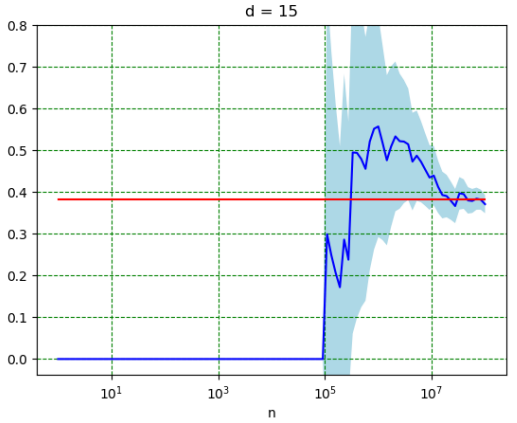

図 15　15 次元球の体積が理論値へと収束していく様子

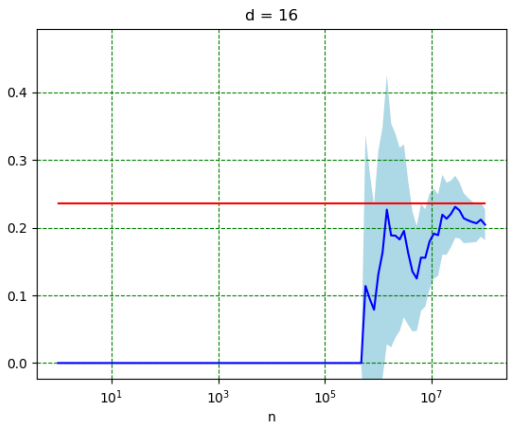

図 16　16 次元球の体積が理論値へと収束していく様子

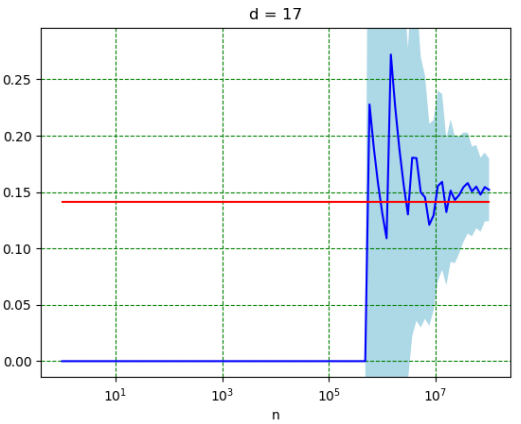

図 17　17 次元球の体積が理論値へと収束していく様子

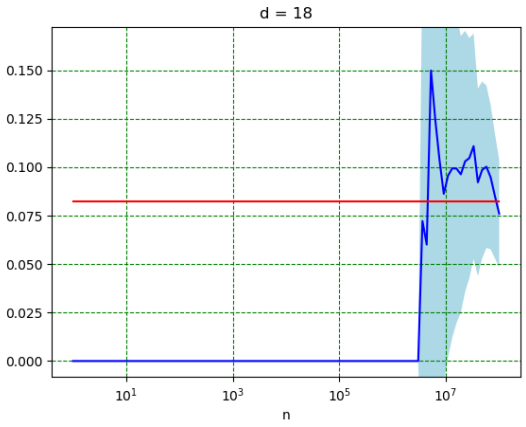

図 18　18 次元球の体積が理論値へと収束していく様子

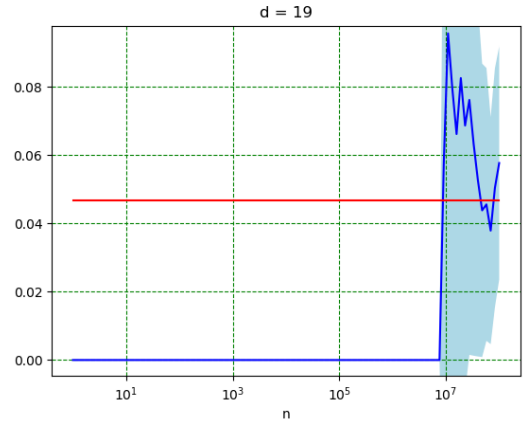

図 19　19 次元球の体積が理論値へと収束していく様子

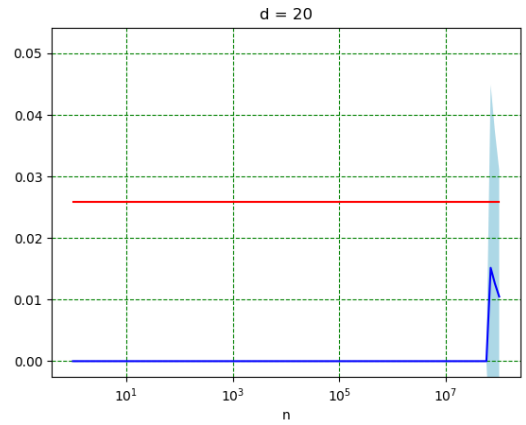

図 20　20 次元球の体積が理論値へと収束していく様子

3.4　考察

誤差は試行回数が大きくなるにつれて小さくなるのが視覚的にわかったが、次元 d が大きくなるにつれて、収束が遅くなり、同じだけ誤差が小さくなるのによりたくさんの点が必要なことがうかがえる。

また、誤差は点が一つ以上球の中に入ってからは、概ね理論値を含むような範囲をとっていることがわかる。

誤差は点が一つ以上球の中に入ってからはいない状態では誤差は 0 になってしまっている。本来はこの状態でも理論値を含むような値[1]をとるべきであるがこうなってしまうのは、試行回数が十分大きいと仮定した上で得られた標本分散をそのまま母分散としてしまっている点にあると思われる。本来は試行回数が小さい場合も考慮して、標本分散から母分散を推定すべきであるが、今回の課題の趣旨とずれるためこれについての作業は行わなかった。機会があったら行いたい。

4　次元を変えたときの超球の体積の変化

4.1　内容

また、次元を変えたときの超球の体積の変化を横軸次元、縦軸体積と誤差 1.96σ としてプロットした。試行回数は 10^8 回行った。

4.2　ソースコード

C++ のコードは 3 と同一である。

plotmulti_v_s.py

```
1  # -*- coding: utf-8 -*-
2
3  import matplotlib.pyplot as plt
4  import numpy as np
5  import csv
6
7  from matplotlib import ticker
8
9
10 def main():
11   data = []
12   with open('../../cpp/out/v_s_plot.csv
         ') as f:
13       rows = csv.reader(f, quoting=csv.
             QUOTE_NONNUMERIC)
14       for row in rows:
15           data.append(row)
16
17   # print(data)
18   d = [row[0] for row in data]
19   v = [row[1] for row in data]
20   error = [row[2] for row in data]
21
```

[1] 例えば無限

```
22    # v_plus_error = [e + error[i] for i,
          e in enumerate(v)]
23    # v_minus_error = [e - error[i] for i,
           e in enumerate(v)]
24
25    # plt.plot('o')
26    # plt.axes(yerr=error).errorbar()
27    # plt.plot(n, v_plus_error, 'b',
          linestyle='solid')
28    # plt.plot(n, v_minus_error, 'b',
          linestyle='solid')
29    plt.errorbar(d, v, yerr=error, fmt='--
          o', ecolor='g')
30
31
32    # plt.axes().set_ylim(ymin=-max(v
          )*0.1, ymax=max(v_plus_error)*1.1)
33    # plt.axes().set_ylim(ymin=-0.1*
          v_theory[0], ymax=2.1*v_theory[0])
34    plt.axes().set_xticks(np.arange(2, 21,
           step=1))
35    # plt.xscale('log')
36    # plt.yscale('log')
37    plt.grid(True, which="both", axis='
          both', ls="--", color="grey")
38    plt.xlabel('d')
39    plt.ylabel('v ^^c3^^82±  1.96 ^^c3^^8
          f^^c6^^92')
40    plt.title('n = 10 ** 8')
41    plt.savefig('out/v_s_plot.eps')
42    plt.savefig('out/v_s_plot.png')
43    plt.show()
44
45
46  if __name__ == "__main__":
47    main()
```

4.3 結果

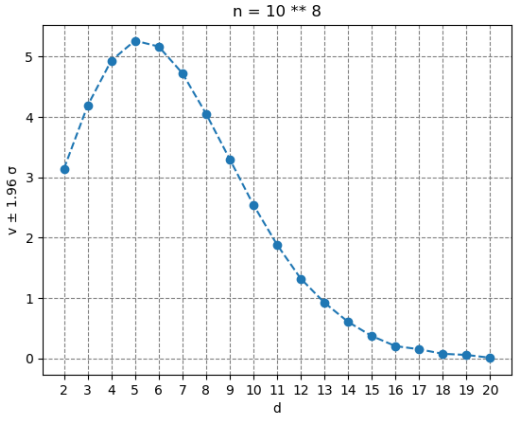

図 21　各次元ごとにおける超球の体積 $V_n(R)$ とその誤差 1.96σ（誤差が小さすぎて見えないが…）

4.4 考察

誤差 1.96σ も縦軸にプロットしているのだが体積は 5 次元のときに最も大きいことがわかる。10^8 回の試行で低次元においてモンテカルロ法で求めた超球の体積が理論値に収束していることは 3 で確かめたし、この図から誤差はかなり小さいことがうかがえるので[2]、このプロットが正確であることはいうまでもないだろう。

参考文献

[1] https://ja.wikipedia.org/wiki/モンテカルロ法
[2] https://ja.wikipedia.org/wiki/超球の体積

[2] というより誤差が小さすぎてそもそもプロットされているかもわからない。