シミュレーション工学　モンテカルロ法

三浦夢生

2020.11.11 Wed

1　モンテカルロ法

モンテカルロ法とは数値計算分野において乱数を用いて行う手法の総称であるが，今回は乱数を用いて面積の近似解を求める方法のことを指して使う．乱数を用いるため，品質の良い疑似乱数生成アルゴリズムを選ぶ必要がある．

ある領域の面積 S を求めたい場合，その領域を含む面積 T の中にランダムに点を打ち，領域内に点が含まれる確率 p を用いて $S = pT$ と書ける．

有名なものでは，モンテカルロ法を用いて円周率を求めるものがある．

2　乱数生成器

2.1　線形合同法

疑似乱数を生成する方法の一つで，以下の式で示される．

$$X_{n+1} = (A * X_n + B) \bmod M \tag{1}$$

ここで X_0 は初期値 (シード値)，A, B, M はパラメータであり，この値のとり方によって疑似乱数の質が変わる．この方法は一般にあまり質が良くないとされる．

2.2　メルセンヌ・ツイスタ

高次元に均等分布する乱数を高速に生成できる非常に評価の高い疑似乱数生成器である．内部状態の大きさや周期は設定可能であるが，基本的に $2^{19937} - 1$ が用いられる．他の疑似乱数生成器がもつ多くの欠点を克服しているため広く利用されている．

3　円周率

3.1　ライプニッツの公式

円周率を求めるための公式の一つで，以下の級数で示される．

$$1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots = \frac{\pi}{4} \tag{2}$$

総和の記号を用いると以下のようにも表せる．

$$\sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} = \frac{\pi}{4} \tag{3}$$

美しくまとまっているが，収束が遅いことで知られる．

3.2 マチンの公式

マチンは $\arctan x$ の関係式を考え，グレゴリー級数と結びつけてより速く収束する級数を得た．それを以下に示す．

$$4\arctan\frac{1}{5}-\arctan\frac{1}{239}=\frac{\pi}{4} \tag{4}$$

計算においては上記の公式の両辺を 4 倍し，$\arctan x$ をグレゴリー級数に直して差を取る．

$$d(m)=16\sum_{n=0}^{3m+2}\frac{(-1)^n}{2n+1}\left(\frac{1}{5}\right)^{2n+1}-4\sum_{n=0}^{m}\frac{(-1)^4}{2n+1}\left(\frac{1}{239}\right)^{2n+1} \tag{5}$$

ここで，第 1 項と第 2 項の項数が違うのは，$\frac{1}{5}$ と $\frac{1}{239}$ で値が大きく異なるため，項全体の値の大きさを近づけるためである．

3.3 ラマヌジャンによる公式

ラマヌジャンはモジュラー関数と呼ばれる考えをもとに以下の式を発見した．

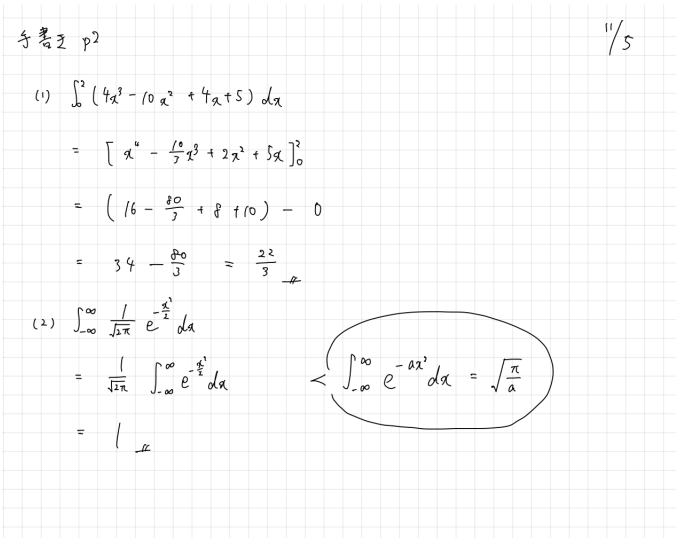
$$\frac{1}{\pi}=\frac{2\sqrt{2}}{99^2}\sum_{n=0}^{\infty}\frac{(4n)!(1103+26390n)}{(4^n99^n n!)^4} \tag{6}$$

この公式は非常に収束が速いことで知られる．

4 課題

4.1 課題 1

積分 2 問の手計算した画像を以下に示す．



4.2 課題 2

上記と同様の問題をモンテカルロ法のプログラムで解いた結果を以下に示す．ただし，問 2 の積分範囲は $-5 \leq x \leq 5$ としている．またプログラムリストを付録に示す．

```
% python3 MonteCarlo1.py
S = 7.33278
```

```
% python3 MonteCarlo2.py
S = 0.70286
```

4.3 課題 3

自作の線形合同法とメルセンヌ・ツイスタで乱数を生成し，円周率を求めた結果を以下に示す．またプログラムリストを付録に示す．

```
% python3 pi2_1.py
pi = 3.14872
```

```
% ./MT_pi
pi = 3.147680
```

後述のプログラムリストの通り，ランダムデータの数が少ないためあまり比較が出来なかった．

4.4 課題 4

先述した 3 つの方法で円周率を求めた結果を以下に示す．またプログラムリストを付録に示す．

```
% python3 pi1.py
n = 1000000
leibniz: 3.1415936535887745
n = 10
machin: 3.141592653589794
n = 1
ramanujan: 3.141592653589794
```

試行回数からわかるように収束の速さに違いが見られる．

5 付録

今回作成したプログラムを以下に示す．

5.1 モンテカルロ法積分プログラム 1

```
import numpy as np

def func(x):
   return 4*(x**3) - 10*(x**2) + 4*x + 5

def generate_random(a, b, maxY, t):
    dataX = []
    dataY = []
    dataX = np.random.rand(t) * (b - a) + a
    dataY = np.random.rand(t) * maxY
    return dataX, dataY

def montecarlo(t, dataX, dataY):
    cnt = 0
    for i in range(len(dataX)):
        if func(dataX[i]) >= dataY[i]:
            cnt += 1
    p = float(cnt) / float(t)
    return p

if __name__ == '__main__':
    t = 1000000
    a = 0
    b = 2
    maxY = 10
    cnt = 0

    dataX, dataY = generate_random(a, b, maxY, t)
    p = montecarlo(t, dataX, dataY)
    print('S =', (b-a) * maxY * p)
```

5.2 モンテカルロ法積分プログラム 2

```
import numpy as np

def func(x):
   return (1/np.sqrt(2*np.pi)) * np.exp(-x**2)

def generate_random(a, b, maxY, t):
    dataX = []
    dataY = []
    dataX = np.random.rand(t) * (b - a) + a
    dataY = np.random.rand(t) * maxY
    return dataX, dataY
```

```
12
13 def montecarlo(t, dataX, dataY):
14     cnt = 0
15     for i in range(len(dataX)):
16         if func(dataX[i]) >= dataY[i]:
17             cnt += 1
18     p = float(cnt) / float(t)
19     return p
20
21 if __name__ == '__main__':
22     t = 1000000
23     a = -5
24     b = 5
25     maxY = 2
26     cnt = 0
27
28     dataX, dataY = generate_random(a, b, maxY, t)
29     p = montecarlo(t, dataX, dataY)
30     print('S =', (b-a) * maxY * p)
```

5.3 線形合同法円周率プログラム

```
1  def generate_point(times):
2      dataX = LCGs(times, 1)
3      dataX = [i / 2**32 for i in dataX]
4      dataY = LCGs(times, 2)
5      dataY = [i / 2**32 for i in dataY]
6      return dataX, dataY
7
8  def LCGs(num, seed):
9      A = 1664525
10     C = 1103515245
11     M = 2**32
12     x = seed
13     r_list = []
14     for i in range(num):
15         x = (A * x + C) % M
16         r_list.append(x)
17     return r_list
18
19 def count_point(dataX, dataY):
20     cnt = 0
21     for x, y in zip(dataX, dataY):
22         if x**2 + y**2 < 1:
23             cnt += 1
```

```
    return cnt

def main():
    times = 100000

    dataX, dataY = generate_point(times)
    cnt = count_point(dataX, dataY)
    print('pi =', 4 * cnt / times)

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

5.4 メルセンヌ・ツイスタ円周率プログラム

```
#include <stdio.h>
#include "MT.h"

#define NUM 100000

int count_point(double dataX[NUM], double dataY[NUM]);

int main(void)
{
        int i;
        int seed;
        int cnt;
        double pi;
        double dataX[NUM];
        double dataY[NUM];

        seed = 0;
        init_genrand(seed);

        for (i = 0; i < NUM; i++) {
                dataX[i] = genrand_real1();
                dataY[i] = genrand_real1();
        }

        cnt = count_point(dataX, dataY);
        pi = 4 * (double)cnt / (double)NUM;

        printf("pi = %lf\n", pi);

        return 0;
}
```

```
32
33 int count_point(double dataX[NUM], double dataY[NUM])
34 {
35         int i;
36         int cnt;
37
38         cnt = 0;
39
40         for (i = 0; i < NUM; i++) {
41                 if (dataX[i] * dataX[i] + dataY[i] * dataY[i] < 1) {
42                         cnt++;
43                 }
44         }
45
46         return cnt;
47 }
```

5.5 ライプニッツ・マチン・ラマヌジャン円周率プログラム

```
1  import math
2
3  def leibniz():
4      n = 1000000
5      print('n =', n)
6      s = 0.0
7      for i in range(n+1):
8          s += (-1)**i / ((2*i) + 1)
9      pi = s * 4
10     return pi
11
12 def machin():
13     n = 10
14     print('n =', n)
15     num_1 = 0.0
16     for i in range(3*n+2+1):
17         num_1 += ((-1)**i / (2*i+1)) * ((1/5)**(2*i+1))
18     num_1 = num_1 * 16
19     num_2 = 0.0
20     for i in range(n+1):
21         num_2 += ((-1)**i / (2*i+1)) * ((1/239)**(2*i+1))
22     num_2 = num_2 * 4
23     pi = num_1 - num_2
24     return pi
25
26 def ramanujan():
```

```
27      n = 1
28      print('n =', n)
29      s = 0.0
30      for i in range(n+1):
31          temp1 = 1
32          for j in reversed(range(1, 4*i+1)):
33              temp1 *= j
34          temp2 = 1
35          for j in reversed(range(1, i+1)):
36              temp2 *= j
37          s += (temp1 * (1103+26390*i)) / (((4**i) * (99**i) * temp2))**4
38      s = s * 2 * math.sqrt(2) / (99**2)
39      pi = 1 / s
40      return pi
41
42  if __name__ == '__main__':
43      pi_1 = leibniz()
44      print('leibniz:', pi_1)
45      pi_2 = machin()
46      print('machin:', pi_2)
47      pi_3 = ramanujan()
48      print('ramanujan:', pi_3)
```