

## 第 14 回 数値計算アルゴリズム (モンテカルロ法)

今回と次回は、数式などの近似解をコンピュータで求める手法を学ぶ。

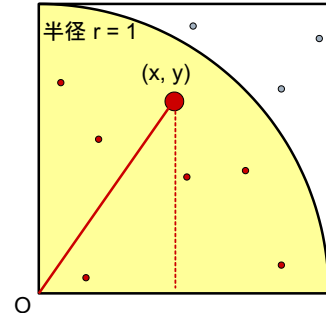
### 【モンテカルロ法】

まずは、シミュレーションにより近似解を求める「モンテカルロ法」について学ぶ。モンテカルロ法は、多くの回数シミュレーションを実行することで、解析的に解くことが困難な問題であっても近似的な解を求める手法である。

モンテカルロ法は物理学、統計学、工学、生物学などの分野で使われる。金融の分野では、コーポレート・ファイナンスや数理ファイナンスの分野で使われる<sup>1</sup>。

ここでは、円周率 $\pi$ をモンテカルロ法により次の手順で求めるプログラムを考える。

- (1) 辺の長さが 1 の正方形を考え、この正方形の内側に  $n$  個の点をランダムに打つ。
- (2) この正方形の左下の点を中心とする半径 1 の円の 1/4 部分を考え、この 1/4 円の中に入った点の数  $\text{count}$  を数える。
- (3)  $\text{count}$  を  $n$  で割った値は、1/4 円の面積  $\pi/4$  の近似値と考えられる。この関係を利用して  $\pi$  を近似的に求める。



### 【★課題】

モンテカルロ法により  $\pi$  を求めるプログラムを????の箇所を修正させることによって完成させて、ToyoNet-ACE から提出してください。モンテカルロシミュレーションで  $\pi$  の近似をするという趣旨のプログラムであるから、 $\text{pi} =$  のところに直接  $\pi$  の近似値を書くような方法ではこの課題を解いたことにはならず、不正解となる。

```
import random
n = int(input())
count = 0
for i in range(n):
    x = random.random()
    y = random.random()
    dist = ??? # (x, y) の原点からの距離の2乗
    if dist <= 1:
        count += 1
pi = ???
print(pi)
```

入力 : 10

出力 : 3.2 (乱数が使われているので、実行するたびに値が変わる)

変数  $\text{dist}$  は原点からの距離ではなくて、その 2 乗としているが、 $\text{dist} \leq 1$  で「距離の 2 乗が 1 以下」つまり「距離が 1 以下」となる。Random モジュールの  $\text{random.random}$  関数は、53 ビット精度の浮動小数点で 0 以上 1 未満の一樣乱数を返す関数である<sup>2</sup>。

<sup>1</sup> [https://en.wikipedia.org/wiki/Monte\\_Carlo\\_methods\\_in\\_finance](https://en.wikipedia.org/wiki/Monte_Carlo_methods_in_finance)

<sup>2</sup> <https://docs.python.org/ja/3/library/random.html>

$\pi = 3.1415926535897932\cdots$  である。入力する値を大きくすると、より精度の高い近似が得られる。入力する値を 10, 100, 1000, 10000,  $\cdots$  と増やしていき、どの程度精度が高くなるかを確認してみよう。

【発展：ガウス・ルジャンドルのアルゴリズム】

ガウス・ルジャンドルのアルゴリズムは、円周率の近似値を計算するためのとても収束が速いアルゴリズムで、円周率を多くの桁数計算するために、実際に使われている。

$$a_0 = 1, b_0 = \frac{1}{\sqrt{2}}, t_0 = \frac{1}{4}, p_0 = 1$$

として、次の反復式を a, b が希望する精度になるまで繰り返す。これは、a と b の算術幾何平均（算術平均と幾何平均を繰り返して作られる数列の極限）を計算していることになる。

$$a_{n+1} = \frac{a_n + b_n}{2}$$

$$b_{n+1} = \sqrt{a_n b_n}$$

$$t_{n+1} = t_n - p_n(a_n - a_{n+1})^2$$

$$p_{n+1} = 2p_n$$

円周率  $\pi$  は、a, b, t を用いて以下のように近似される。

$$\pi \approx \frac{(a + b)^2}{4t}$$

★円周率の近似値を計算するプログラムを実行してみよう。

プログラムは [https://paiza.io/projects/gEb4\\_cT5b\\_Xxl03RnCtnnw](https://paiza.io/projects/gEb4_cT5b_Xxl03RnCtnnw)

```
import math
a = p = 1
b = 1 / math.sqrt(2)
t = 0.25
for n in range(5):
    pi = (a + b) ** 2 / (4 * t)
    print("n = {0}: π = {1}".format(n, pi))
    next_a = (a + b) / 2
    b = math.sqrt(a * b)
    t -= p * (a - next_a)**2
    p *= 2
    a = next_a
```