

# cp-15. 疑似乱数と シミュレーション

(C プログラミング入門)

URL: <https://www.kkaneko.jp/pro/adp/index.html>

金子邦彦



# 内容

例題 1 . 疑似乱数

例題 2 . ランダムウォーク

例題 3 . じゃんけんゲーム

例題 4 . モンテカルロ法による数値積分

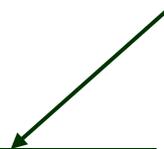
- 疑似乱数を使ったプログラムを理解する
- 疑似乱数を使ったシミュレーションを理解する

# 例題 1 . 疑似乱数

- 疑似乱数を表示するプログラムを作る.
  - 疑似乱数の表示を, 10回繰り返すこと
  - 疑似乱数を発生させるために, srand 関数と rand 関数を使うこと.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#pragma warning(disable:4996)
int main()
{
    int i;
    int x;
    srand( (unsigned int) time(NULL) );
    for ( i = 0; i < 10; i++ ) {
        x = ( (double)rand() / (RAND_MAX+1) ) * 10;
        printf( "x=%d¥n", x );
    }
    return 0;
}
```

疑似乱数のシード  
の設定



疑似乱数の発生



# 実行結果例

x=3

x=1

x=8

x=8

x=4

x=3

x=4

x=1

x=6

x=1

# 疑似乱数

```
srand( (unsigned int) time(NULL) );
```

```
i = 0
```

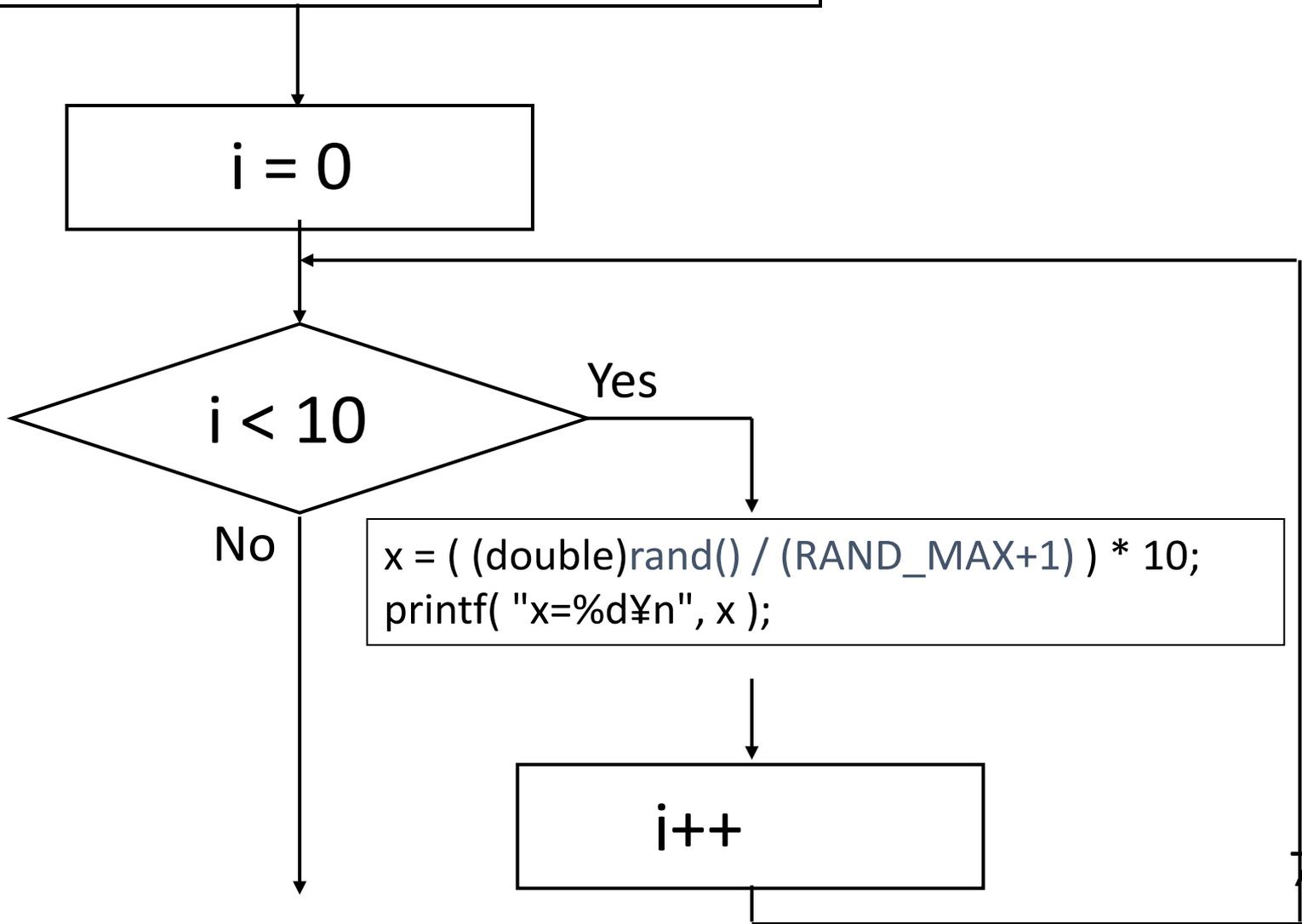
$i < 10$

Yes

No

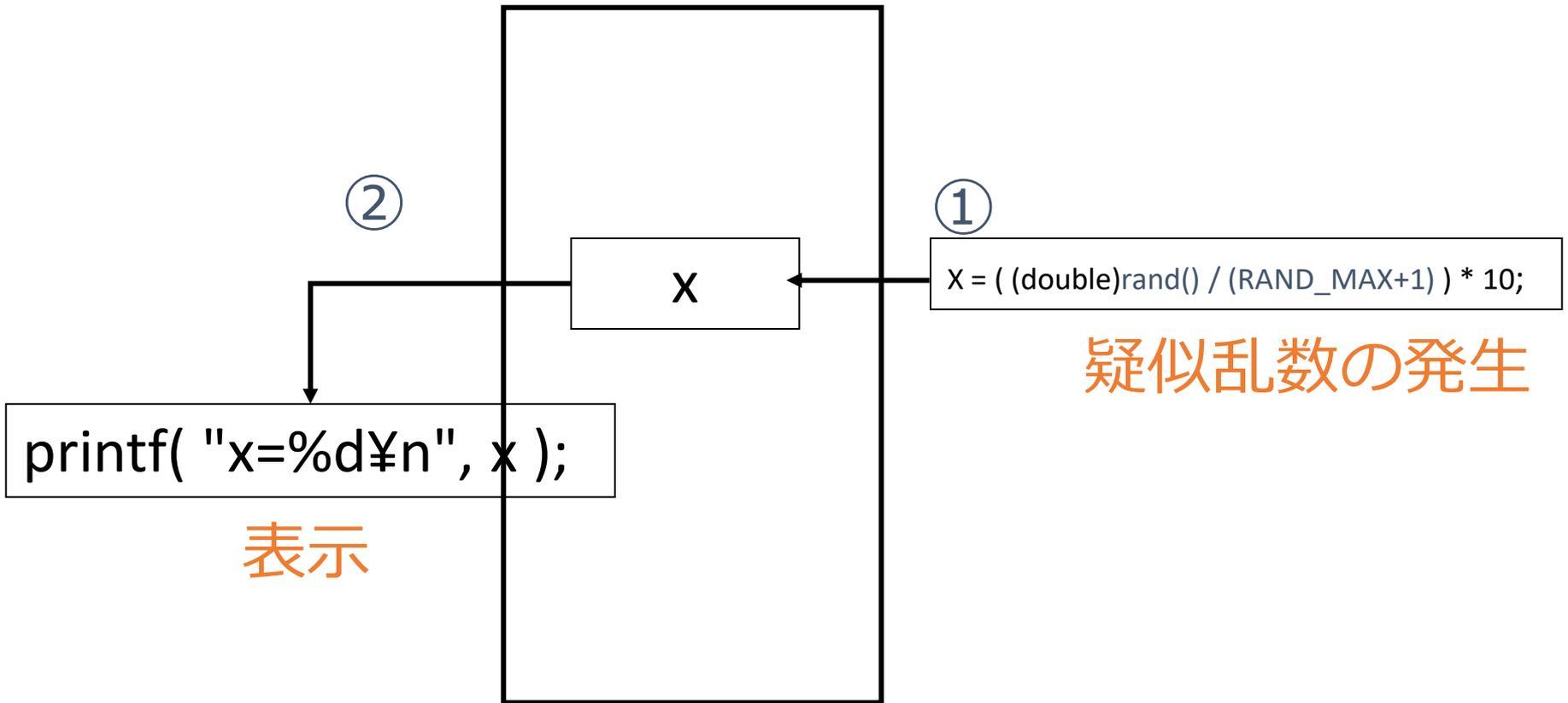
```
x = ( (double)rand() / (RAND_MAX+1) ) * 10;  
printf( "x=%d¥n", x );
```

```
i++
```



# プログラムとデータ

## メモリ



# 疑似乱数



- rand 関数は疑似乱数 (pseudo-random number) を発生させるためのライブラリ関数.
- 発生される数は, 0 からRAND\_MAXの間の値をとる.
  - RAND\_MAX は, rand 関数で発生する疑似乱数 (pseudo-random number) の最大値を表す

# 疑似乱数のシード (seed)



- srand 関数は, rand 関数で発生させる疑似乱数 (pseudo-random number) の系列を設定するためのライブラリ関数.
- 疑似乱数の系列は, srand 関数の引数 seed によって変化する.
- rand 関数は, シードの設定を行わないと, 同じ系列の疑似乱数を返す.

# 疑似乱数のまとめ

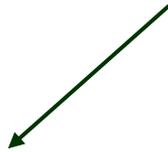
- srand 関数, rand 関数の使用では, `#include <stdlib.h>` が必要
- rand 関数: 疑似乱数の発生
  - 疑似乱数の範囲: 0からRAND\_MAX
  - 疑似乱数の型: 整数データ
  - rand関数を実行するたびに, 新しい疑似乱数が返される
- srand 関数
  - rand 関数は, ある決められた初期値(「シード」という)から, 疑似乱数を計算する
  - プログラムの実行のたびに, シードを変えて, 違う疑似乱数を発生させるために, srand 関数を用いる

## 例題 2. ランダムウォーク

- ランダムウォークのプログラムを作る。
  - 「酔っ払い」が歩いている
    - 「酔っ払い」には記憶がない
    - 「酔っ払い」は確率 0.5 で右に, 確率 0.5 で左に歩く
  - 道の幅は 11 メートル, 1 歩は 1 メートルとし, 最初, 酔っ払いは道の中央にいる. 道幅を超えたら終わり

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#pragma warning(disable:4996)
void print( int n )
{
    int i;
    for ( i = 0; i < n; i++ ) {
        printf( " " );
    }
    printf( "*¥n" );
}
int main()
{
    int n = 5;
    srand( (unsigned int) time(NULL) );
    while ( ( n >= 0 ) && ( n <= 10 ) ) {
        print( n );
        if ( ( (double)rand() / (RAND_MAX+1) ) < 0.5 ) {
            n++;
        } else {
            n--;
        }
    }
    return 0;
}
```

## 疑似乱数のシード の設定



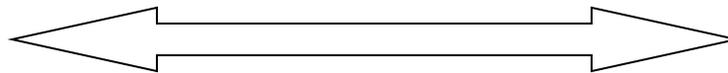
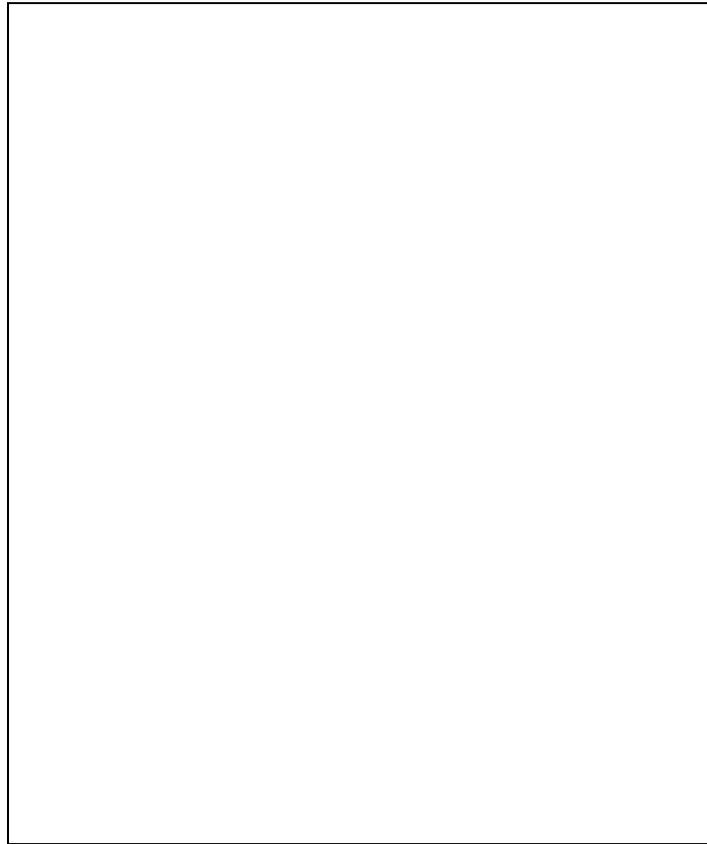
## 疑似乱数の発生



# 実行結果例

```
*  
  
 *  
  
  *  
  
   *  
  
    *  
  
     *  
  
      *  
  
       *  
  
        *  
  
         *
```

0 1 2 3 4 5 6 7 8 9 10



道幅 11メートル

# 課題 1. ランダムウォーク結果集計

- 例題 2 の「ランダムウォーク」を 1000 回繰り返して, 「平均で何歩歩いたかを求めるプログラム」を作りなさい
  - 「小数付きのデータ」を扱うために、浮動小数 (double) を使うこと
  - 各繰り返しにおいて 「n = 5;」 を実行すること

## 例題 3. じゃんけんゲーム

- じゃんけんを行うプログラム

0 : パー

1 : グー

2 : チョキ

じゃんけんの勝負の判定のために、2次元配列を用いる

## 疑似乱数のシード の設定

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#pragma warning(disable:4996)
int main()
{
    int x;
    int y;
    int hantei[3][3] = { {0, 1, -1}, {-1, 0, 1}, {1, -1, 0} };
    char jk[3][20] = { "パー", "グー", "チョキ" };
    srand( (unsigned int) time(NULL) );
```

## 疑似乱数の発生

```
do {
    y = ( (double) rand() / (RAND_MAX+1) ) * 3;
    printf( "¥n" );
    printf( "じゃんけん (0:%s, 1 : %s, 2 : %s, 3 : やめる)¥n", jk[0], jk[1], jk[2] );
    scanf( "%d", &x );
    switch ( hantei[x][y] ) {
        case 1:
            printf( "あなた : %s, 私 : %s, あなたの勝ち！うう悔しい¥n", jk[x], jk[y] );
            break;
        case 0:
            printf( "あなた : %s, 私 : %s, ひきわけ. もう1度勝負！¥n", jk[x], jk[y] );
            break;
        case -1:
            printf( "あなた : %s, 私 : %s, 私の勝ち！やったあ¥n", jk[x], jk[y] );
            break;
    }
} while ( x!= 3 );
return 0;
}
```

## 課題 2. じゃんけん結果集計

- 例題 3 の「じゃんけんプログラム」について、入力されたパー、グー、チョキの回数に関する情報を表示するプログラムを作りなさい
  1. パーの次にパー
  2. パーの次にグー
  3. パーの次にチョキ
  4. グーの次にパー
  5. グーの次にグー
  6. グーの次にチョキ
  7. チョキの次にパー
  8. チョキの次にグー
  9. チョキの次にチョキ

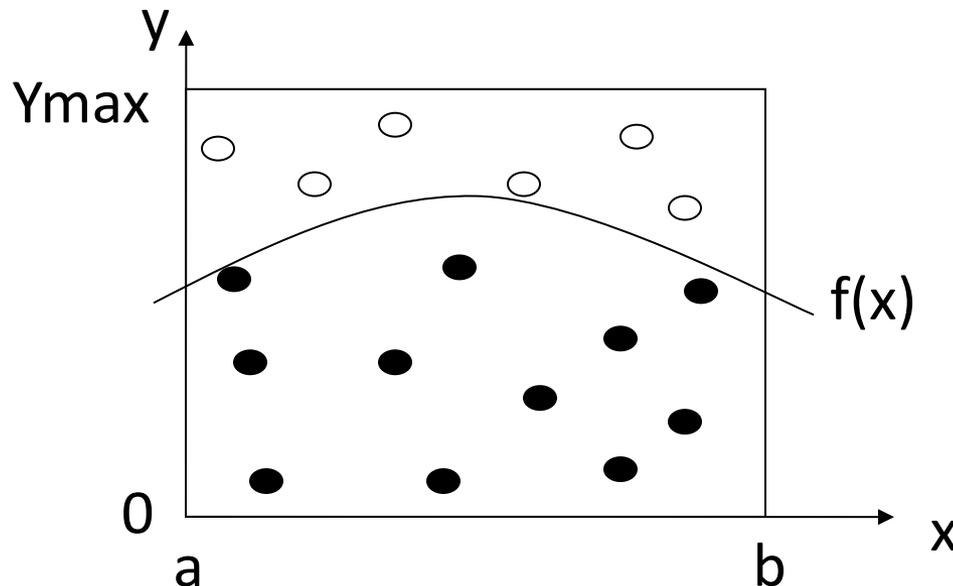
# 例題4. モンテカルロ法による 数値積分



- モンテカルロ法による数値積分を行うプログラムを書く
- 次の値を読み込むこと
  - 積分区間[a,b]
  - 疑似乱数の発生回数
- 数値積分を行うべき  $f(x)$  は, 指数関数  $\exp(-x)$  とする (プログラム中に書く)

# モンテカルロ法とは

- モンテカルロ法は，疑似乱数を用いて積分近似値を求める
- 積分区間 $[a,b]$ で  $f(x)$  の値が0以上で，かつある値 $Y_{max}$ 以下であることが分かっているとき(下図)，
  1. ランダムな座標の発生 (疑似乱数を利用)  
 $a \leq x \leq b, 0 \leq y \leq Y_{max}$ の範囲内でランダムな座標 $(x,y)$ を発生
  2. 関数  $f(x)$  以下であるのかの判定  
関数 $f(x)$ 以下である座標が総数に占める割合を求める以上で，**矩形の面積 $(b-a) \cdot Y_{max}$ に乗じることで積分近似値を求める**



```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
```

```
double f(double x)
{
    return exp(-x*x);
}
```

```
main()
```

```
{  
    double Ymax = 1.0;  
    double a, b, x, y, S;  
    int i, N, seed, count;
```



```
printf("積分区間(a~b) : ");
```

```
printf("a= ? ");
```

```
scanf("%lf", &a);
```

```
printf("b= ? ");
```

```
scanf("%lf", &b);
```

```
printf("疑似乱数の発生回数 N : ");
```

```
printf("N= ? ");
```

```
scanf("%d", &N);
```

積分区間 a, b の読み込み

疑似乱数の発生回数  
の読み込み

```
srand( (unsigned int) time(NULL) );
```

```
count = 0;  
for(i = 0 ; i < N ; i++){
```

```
    x = (double)rand() / (RAND_MAX+1) * (b - a) + a;  
    y = (double)rand() / (RAND_MAX+1) * Ymax;
```

```
    if(y < f(x)) {  
        count++;  
    }
```

} 関数  $f(x)$  以下であるのかの判定

```
    S = (b - a) * Ymax * count / N;
```

```
    printf("面積 = %lf¥n", S);
```

```
}
```

積分値の計算

ランダムな座標値  
(x, y) の発生

# 実行結果の例

$$f(x) = \exp(-x^2)$$

積分範囲(a~b) : 0 1

乱数の個数 : 1000

乱数の種 : 0

面積 = 0.761000

積分範囲(a~b) : 0 1

乱数の個数 : 1000000000

乱数の種 : 0

面積 = 0.746825

積分範囲(a~b) : 0 1

乱数の個数 : 1000000

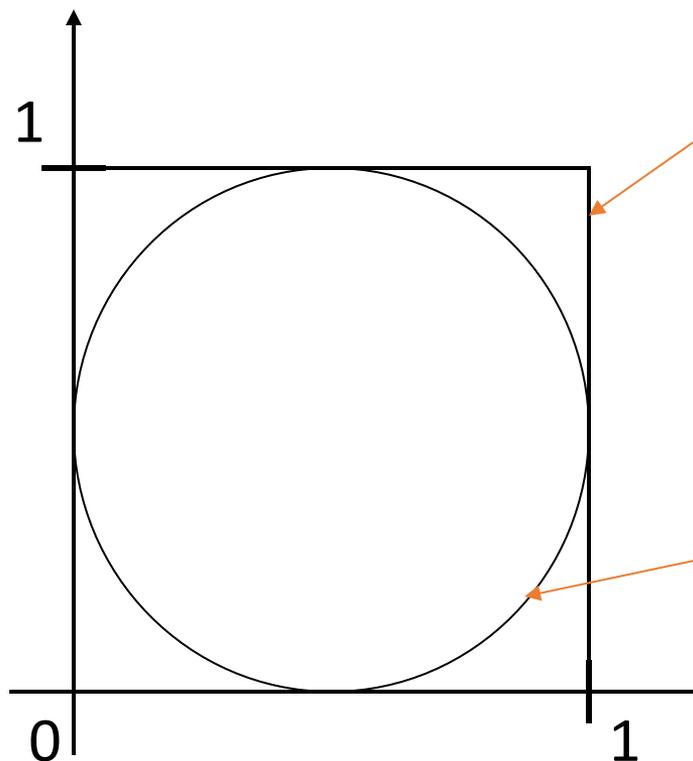
乱数の種 : 0

面積 = 0.747537

## 課題 3. 円周率の計算

モンテカルロ法を用いて, 円周率を求めなさい

# 課題 3 のヒント



① ランダムな座標値  $(x, y)$  をこの正方形内で発生させる

② 発生させた座標値がこの円内かどうかを判定する

(円内である確率は、 $\pi/4$ )