1　円周軌道のダイナミクスのシミュレーション

1.1　シミュレーションの内容とプログラム

円の軌跡を描く次のようなダイナミクスをシミュレートするプログラムを考える．

$$\frac{dx}{dt} = -y, \quad \frac{dy}{dt} = x$$

このダイナミクスを離散化すると次のような式となる．

$$\Delta x = -y \cdot \Delta t, \quad \Delta y = x \cdot \Delta t$$

これを実現するプログラムは，

```
x = x - y*dt,   y = y + x*dt
```

となり，x,y に適当なる初期値を与えてこのプログラムを繰り返すとダイナミクスのシミュレーションが実現できる．これを Python で記述したものが circle.py，C 言語で記述したものが circle.c である．

これらのプログラムは $(x, y) = (1.0, 0.0)$ を初期値として時間の刻み幅 dt を 10^{-8} として円の軌跡をシミュレートする．軌跡が円周を一周するとシミュレーションが終了する．

Python のプログラム：circle.py

```
# coding: utf-8

# 必要なモジュールの読込み
import time

# 対象の関数：円軌道の精密シミュレーション
def calcf(name):
    dt = 0.00000001
    x = 1.0;    y = 0.0
    t1 = time.time()
    for i in range(628318531):
        x -= y * dt;    y += x * dt
    t = time.time() - t1
    print(name,'> time:',t)
    print( '(x,y)=(',x,',',y,')' )

# 実行
if __name__ == '__main__':
    calcf('test')
```

C のプログラム：circle.c

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

int main()
{
    double  x, y, dt;   /* 座標と微小時間 */
    long    c;          /* ループカウンタ */
    clock_t t1, t2;     /* 時間計測用変数 */

    /* シミュレーションの初期設定 */
    x = 1.0;    y = 0.0;
    dt = 0.00000001;    /* 時間間隔 */
    t1 = clock();   /* 開始時刻 */
    for ( c = 0L; c < 628318531L; c += 1L ) {
        y += x * dt;
        x -= y * dt;
    }
    t2 = clock();   /* 終了時刻 */
    printf("(x,y)=(%lf,%lf)\n",x,y);

```

```
21      /* 計算時間の表示（標準エラー） */
22  /*  fprintf(stderr,"%lf (sec)\n", (double)(t2-t1)/1000000.0);    */
23      fprintf(stderr,"%lf (sec)\n", (double)(t2-t1)/1000.0);
24  }
```

※ C のプログラム 22,23 行目の記述は，Windows と Mac で採取時間の単位が違う（ミリ秒とマイクロ秒）ため，どちらかをコメントアウトして実験するためのものである．

ここに示したプログラムは同じアルゴリズムをそれぞれ異なる言語で実装したものであり，Python と C の実行速度の比較をするための 1 つの試みとなる．シミュレーションは double 精度の積を含む演算から成り，(x, y) の移動処理を 6 億回以上繰り返すものである．

1.2 実行結果と評価

先のプログラム circle.py を Python インタプリタで，circle.c を GNU の C コンパイラで翻訳して実行した結果を示す．実行環境としては Apple の Mac と Windows10 の 2 種類を用いて，それぞれの実行環境で Python と C の実行時間の比較を行った．

1.2.1 実行環境 1

OS： macOS Sierra

CPU： Intel Core i5-4278U 2.60GHz

表 1: 実行に要した時間の比較 (1)

C (Xcode 8.3.3)	Python 3.6.2	C に対する Python の実行時間の比
3.588（秒）	97.575（秒）	27.197（倍）

1.2.2 実行環境 2

OS： Windows10

CPU： Intel Core i7-5500U 2.39GHz

表 2: 実行に要した時間の比較 (2)

C (GCC 7.2.0) (MinGW64)	Python 3.6.2	C に対する Python の実行時間の比
2.614（秒）	67.626（秒）	25.871（倍）

上の表に示す実行時間は，プログラムを 3 回実行した結果の計測値の平均値である．

【評価】

Python は C に比べてプログラムの実行に 25 倍以上の時間がかかることがわかる．

1.2.3 Python と Cython の比較

circle.py と同様の Cython のプログラム circleC2.pyx を次に示す．

Cython のプログラム：circleC2.pyx

```
1  # coding: utf-8
2
3  # 必要なモジュールの読込み
4  import time
5
6  # 対象の関数：円軌道の精密シミュレーション
7  def calcf(name):
```

```
 8      cdef double dt, x, y     # 型宣言
 9      cdef long   i            # 型宣言
10      dt = 0.00000001
11      x = 1.0;     y = 0.0
12      t1 = time.time()
13      for i in range(628318531):
14          x -= y * dt;     y += x * dt
15      t = time.time() - t1
16      print(name,'> time:',t)
17      print( '(x,y)=(',x,',',y,')' )
18
19  # 実行
20  if __name__ == '__main__':
21      calcf('test')
```

解説： 8〜9 行目で変数の型を宣言している.

このプログラムを Cython 処理系で翻訳して Python 用のモジュールを作成して実行する．実行時間の比較を表 3 に示す.

表 3: Python と Cython の実行時間の比較

Cython 0.26.1	Python 3.6.2	Cython に対する Python の実行時間の比
2.676（秒）	67.626（秒）	25.271（倍）

今回の実験では，Cython による実行時間が C 言語の場合とほぼ同等であることがわかる.