データサイエンス系科目群（理・博）：シミュレーション実習
第 1 回 講義資料

担当：川﨑猛史

名古屋大学大学院理学研究科理学専攻物理科学系・非平衡物理研究室 (R 研)

Last update: April 16, 2023

目次

目次

1.1. 実習担当者と実習方針

- 担当者：川﨑猛史（理学研究科理学専攻物理科学領域　非平衡物理学研究室）
  email:kawasaki@r.phys.nagoya-u.ac.jp
- 講義の進め方：講義資料を TACT から配布＋対面実習
- 参考書
  1. Python コンピュータシミュレーション入門（橋本洋志・牧野浩二著，オーム社）
  2. やさしい C，やさしい C ++（高橋麻奈著，ソフトバンク）
  3. C 言語によるプログラミングの基礎（田中敏幸著，コロナ社）
  4. コンピュータシミュレーションの基礎（岡崎進・吉井範行著，化学同人）
  5. D. Frenkel and B. Smit, Understanding Molecular Simulation: From Algorithms to Applications (Elsevier, 2001) [1].
  6. M. P. Allen and D. J. Tildesley, Computer Simulation of Liquids: Second Edition (Oxford University Press, 2017) [2].
- 成績:
  - 出欠アンケート **30** %（毎回小テストより）＋中間レポート課題 **30** %＋期末レポート課題 **40** %

1.2. 本実習の目的と注意点

目的：

- 理学研究のあらゆる場面で必須となる計算機シミュレーションの実践的な知識および技能の修得を目的とする．
- 目標達成の為に，C（C++）や Python を用いた解析や作図に関する初歩的なプログラミングから，モンテカルロ法や分子動力学法に至る実践的な数値計算手法に関する，理学研究への幅広い応用を意識した講義および実習を行う．

注意点：

- 毎回の出欠アンケートを（TACT 小テストより）実施する．質問や感想を自由に記述してください．
- 毎回の実習用自主課題を出題する．
- 上とは別に，レポート課題を 2 回出題する（TACT 課題より）．
- 本実習では主な計算を C（C++），作図は Python を用いたプログラム例を挙げ，内容の説明を与える．
- ただし，自主課題やレポート課題は，他の言語で行なってもよいし，また棲み分けを自由に変えてもよい（全て Python で書くなど）．

サンプルプログラム等：

- 以下の Github リポジトリに格納しますので適宜取得してください．[リンク][3]

1.3. 講義のスケジュール

- 当実習は春 1 期にて実施する.
- 講義資料は各講義予定日当日朝 **11** 時迄にアップロードする.
- スケジュール
  1. **4/17**：第 **1** 回
  2. 4/24：第 2 回
  3. 5/01：第 3 回
  4. 5/08：第 4 回 (中間レポート課題公開)
  5. 5/15：第 5 回
  6. 5/22：第 6 回 (中間レポート課題提出期限予定)
  7. 5/29：第 7 回
  8. 6/05：第 8 回 (期末レポート課題公開)
  9. 6/12：第 9 回　（補講）

1.4. シラバス

当実習では以下の内容を扱う予定である（進捗に合わせ変更する可能性がある）.

1. 導入
   - C(C++) の使い方 (主に数値計算)
   - Python の使い方 (データ解析と作図)
   - 数値計算の理念
   - 桁落ち
   - 科学計算における無次元化
2. 常微分方程式の数値解法：減衰振動や調和振動子を例に
   - 微分方程式の数値積分
   - 軌道の安定性と保存則
3. １粒子系のブラウン運動
   - ランジュバン方程式（確率微分方程式）
   - 正規乱数の生成法
   - オイラー・丸山法
   - 時間平均とアンサンブル平均
4. 多粒子系のブラウン運動
   - 相互作用力の計算方法
   - 非平衡系のシミュレーション：相分離現象を例に
5. 多粒子系の分子動力学シミュレーション
   - 位置ベルレ法と速度ベルレ法
   - 多粒子系における保存則
6. モンテカルロ法
   - 統計力学の復習
   - マルコフ連鎖モンテカルロ法
   - メトロポリス判定法

目次

2.1. C(C++) の環境構築

- 本実習では，主な数値計算を C(C++)，作図を python で行う．
- C(C++) 環境構築について：
  - **Windows:** MinGW(Minimalist GNU for Windows) のインストール [R 研浅谷氏・池田氏作成マニュアル：リンク]
  - **mac:** homebrew のインストール [**homebrew** 本家リンク][4]
    コンパイラ gcc のインストール [**gcc** インストール方法] [5]

リスト 1: （端末上） homebrew を用いた gcc のインストール．

```
1 brew install gcc
```

2.2. C(C++) プログラムのコンパイル

以下の C(C++) プログラム (pi.cpp) を自身のコンピュータでコンパイルし実行する方法を説明する．

- **C コンパイラ:**

様々なコンパイラー

- gcc:　GNU C compiler collection.
- g++:　GNU C++ compiler collection.
- clang: gcc の代替品 – macOS の標準コンパイラ.
- clang++:　g++の代替品 – macOS の標準コンパイラ．
- icc:　intel c/c++ compiler – 高価だが計算スピードは速い．

コンパイルオプション

- -O3:　One of the optimization options (i.e., -O -O0 -O1 -O2 -O3 -Os -Ofast -Og). -O3 is most used.
- -o:　Specifies the name of the output file.

- サンプルプログラム “pi.cpp”

2.2. C(C++) プログラムのコンパイル (2)

リスト 2: サンプルプログラム “pi.cpp”

```
1  #include <stdio.h> // for printf, etc
2  #include <stdlib.h> // for rand(), etc
3  #include <math.h> // for sin(),cos(), etc
4  #include <iostream>// for cout, etc
5  #include <iomanip>// for setprecision()
6  #include <fstream> // for ifstream/ofstream
7  #include <time.h>// for time(NULL), etc
8
9  int main(void){
10   int i, count = 0, max = 1e+5;
11   double x,y,z,pi;
12   char fname[128];
13   std::ofstream file;
14   srand(time(NULL));  // "time(NULL)" as a seed of ramdom number
15   sprintf(fname,"coord%d.dat",max); // Define the file name for fname[128]
16   file.open(fname); // "file" with the name of fname[128]
17   for(i=0;i<max;i++){
18     x = (double)rand()/RAND_MAX;
19     y = (double)rand()/RAND_MAX;
20     z = x*x + y*y;
21     if(z<1){
22       count++;
```

2.2. C(C++) プログラムのコンパイル (3)

```
23        file << x <<" "<<y <<std::endl;
24      }
25    }
26      file.close();
27      pi = (double)count / max * 4.;
28      printf("%.20f\n",pi); //by C, %.20f -- Displaying with 20 decimal precision
29      std::cout<< std::setprecision(21) <<  pi  << std::endl; // by C++
30
31      return 0;
32  }
```

- 指定 GitHub リポジトリ [リンク][3] からプログラムをダウンロード．
- GitHub からのダウンロード方法 [リンク][6]．
- ダウンロードできらたら，適当なディレクトリー内に pi.cpp を配置する．
- pi.cpp が配置されたディレクトリー内で以下のコマンドを実行する．
- コンパイル方法:

2.2. C(C++) プログラムのコンパイル (4)

リスト 3: （端末上） コードのコンパイル例（いずれかを実行）. -o オプションで実行ファイルに名前をつける．デフォルトでは a.out になる．

```
1 g++ -O3 pi.cpp
2 g++ -O3 pi.cpp -o pi.out
3 g++ -O3 -o pi.out  pi.cpp
4 clang++ -O3 -o pi.out  pi.cpp
```

- 実行方法 (Unix と windows で仕様が異なる)：

リスト 4: (mac や linux 端末上) プログラム (pi.out) の実行．

```
1 ./pi.out
```

リスト 5: (windows 端末上) プログラム (pi.out) の実行． “./”は不要．

```
1 pi.out
```

- 実行結果:

2.2. C(C++) プログラムのコンパイル (5)

リスト 6: (端末上) プログラムの実行結果例．値は乱数の種を時刻にしているため毎回変わることに注意．

```
3.14536000000000015575
3.14536000000000015575
```

便利なエディター （任意）： Visual Studio Code

設定マニュアル（R 研池田氏作成）：

- Win: [リンク]
- Mac:　[リンク]

2.3. Python 環境の構築

- 本実習では Python のプログラムを Jupyter Notebook を用いて実行する．
- Jupyter Notebook は Anaconda Navigator（[本家リンク][7] 図 **1**）の中に格納されている．
- Anaconda Navigator のインストール方法は以下を参照せよ．
  - **Windows/mac:** Anaconda Navigator のインストール [**windows** マニュアル] [**mac** マニュアル]
- Jupyter note book のインストール方法は以下の図 **2** に示した．

2.3. Python 環境の構築 (2)

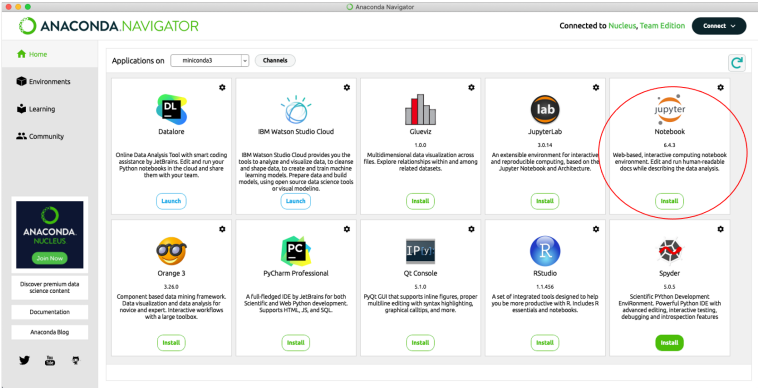
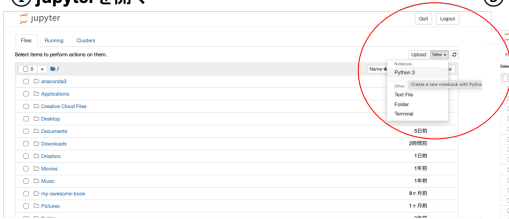


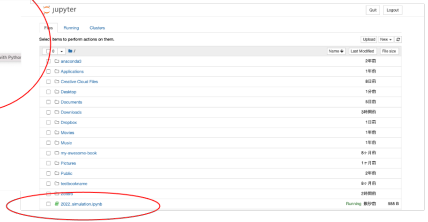
図 1: Anaconda Navigator. この中にある Jupyter note book(赤丸) をインストールし実行．これで python が使えるようになる．

2.3. Python 環境の構築 (3)

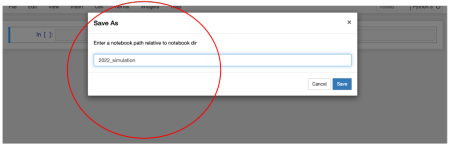
① jupyterを開く



③ ②で作ったnotebookを開く



②Newのタブを開きpython3を選び，新規Notebookに名前をつける．



④ グレーの箱にプログラムを書き込み，shift+enterで実行

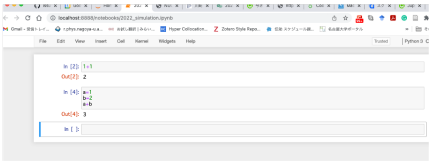


図 2: Jupyter note book 起動後から python 実行まで．

2.4. Python (matplotlib) の実行

- Jupyter Notebook 上で以下のプログラム (matplotlib.pyplot による作図) を実行せよ (実行結果図 **3**).
- （参考）matplotlib のマニュアル [リンク][8]

Python を用いた作図に関する補足 [9]

- matplotlib：Python のグラフィックパッケージ.
- matplotlib．pyplot: axes, figure, plot, scatter など多数のグラフツールを提供.

リスト 7: matplotlib．pyplot のインポートと使用例

```
1 import matplotlib.pyplot as plt
2 fig = plt.figure(figsize=(7,7))
```

- NumPy: 高速演算を提供．多くのパッケージが NumPy を参照．多彩な線形代数演算・統計演算・特殊関数などを提供.

リスト 8: NumPy のインポート

```
1 import numpy as np
```

2.4. Python (matplotlib) の実行 (2)

リスト 9: Python サンプルプログラム “matplot.py”

```
1  import matplotlib.pyplot as plt
2  %matplotlib inline
3  # グラフの解像度が上がる
4  %config InlineBackend.figure_format = 'retina'
5  price = [100, 250, 380, 500, 700]
6  number = [1, 2, 3, 4, 5]
7
8  # グラフを書く
9  plt.plot(price, number)
10
11 # グラフのタイトル
12 plt.title("price / number")
13
14 # 軸のラベルx
15 plt.xlabel("price")
16
17 # 軸のラベルy
18 plt.ylabel("number")
19
20 # 表示する
21 plt.show()
```

2.4. Python (matplotlib) の実行 (3)

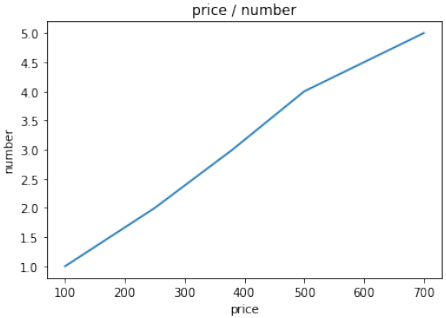


図 3: matplot.py の実行結果．

- 次に，リスト 2 のプログラムの実行で吐き出された “coord100000.dat”の散布図を以下のように作成してみよ．
- サンプルプログラム “coord.py”では，少し凝ったことをしている．図の大きさ，アスペクト比，軸の太さ，字の大きさ，フォント（Tex）などを用いた．実行結果 (図 **4**)

2.4. Python (matplotlib) の実行 (4)

リスト 10: Python サンプルプログラム “coord.py”

```
1  import matplotlib
2  import matplotlib.pyplot as plt
3  %matplotlib inline
4  #図の解像度が上がる
5  %config InlineBackend.figure_format = 'retina'
6  import numpy as np
7  #フォントTex
8  #plt.rcParams["text.usetex"] =True
9
10 #図全体のサイズやアスペクト比を変える
11 fig = plt.figure(figsize=(7,7))
12
13 #複数の図を並べる時ここを変える
14 ax = fig.add_subplot(111)
15
16 #各自ファイルのパスを変えること
17 x, y  = np.loadtxt("./Lecture1/coord100000.dat", comments='#', unpack=True)
18 plt.plot(x, y, "o",markersize=0.5,color="b",label=r"$x^2+y^2\leq1$")
19
20 #図の書式設定
21 plt.tick_params(which='major',width = 1, length = 10)
22 plt.tick_params(which='minor',width = 1, length = 5)
```

2.4. Python (matplotlib) の実行 (5)

```
23 ax.spines['top'].set_linewidth(3)
24 ax.spines['bottom'].set_linewidth(3)
25 ax.spines['left'].set_linewidth(3)
26 ax.spines['right'].set_linewidth(3)
27 plt.xlabel(r"$x$",color='k', size=30)
28 plt.ylabel(r"$y$",color='k', size=30)
29 plt.xticks(color='k', size=25)
30 plt.yticks(color='k', size=25)
31 #図の凡例の有無や位置，サイズを調整
32 plt.legend(ncol=1, loc=1, borderaxespad=0, fontsize=25,frameon=True)
33 #図のマージン設定
34 plt.subplots_adjust(wspace=0.0, hspace=0.25)
35 #各グラフのアスペクト比をにする1:1
36 ax.set_aspect('equal', adjustable='box')
37 #各自ファイルのパスを変えること.
38 plt.savefig('./Lecture1/coord.png')
39 plt.savefig('./Lecture1/coord.pdf')
```

2.4. Python (matplotlib) の実行 (6)

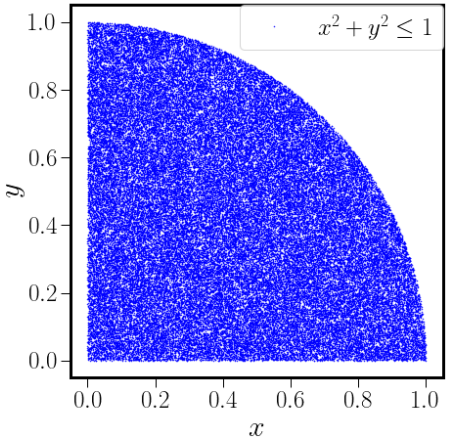


図 4: coord.py の実行結果．

目次

3. 第 1 回自主課題

π の収束性（モンテカルロシミュレーション）

(1) リスト 2 のサンプルプログラムに関して，乱数の発生回数を変化させた際の円周率の収束性について考える．乱数の発生回数 n に対する数値計算上の円周率 $\pi(n)$ を様々とることで，横軸に n，縦軸に $\pi(n)$ の関係を作図せよ．また，理論値 π との誤差 $\delta(n) = |\pi(n) - \pi|$ が n に対してどのように変化するか図示し評価せよ．

(2) （発展問題）大規模な数値計算においては時として，擬似乱数生成器 rand() では乱数周期が短いため不具合が生じることがある．一方，高品質な擬似乱数生成器として松本眞氏により開発されたメルセンヌツイスタが有名である [参考リンク][11]．いま，リスト 2 のサンプルプログラムを，メルセンヌツイスタを用いたものに書き換えよ [参考リンク][12]．指定 GitHub リポジトリ [リンク][3] に，メルセンヌツイスタのヘッダファイルを置いたのでこれを用いてもよい．

目次

参考文献・ウェブサイト

[1] Daan Frenkel and B. Smit.
Understanding Molecular Simulation: From Algorithms to Applications.
Elsevier, October 2001.

[2] Michael P. Allen and Dominic J. Tildesley.
Computer Simulation of Liquids: Second Edition.
Oxford University Press, June 2017.

[3] Takeshi Kawasaki.
2023-simulation-training (GitHub), April 2022.

[4] Homebrew (The Missing Package Manager for macOS (or Linux)).
https://brew.sh/index_ja.

[5] Daiki Suyama.
Mac で GCC を”正しく”環境構築しよう！
https://qiita.com/DaikiSuyama/items/09f5aa399aad37783146.

[6] GitHub からファイルをダウンロードする方法【2020 年 12 月追記】.
https://tetsufuku-blog.com/github-download/, April 2020.

[7] Anaconda Navigator — Anaconda documentation.
https://docs.anaconda.com/anaconda/navigator/index.html.

参考文献・ウェブサイト (2)

[8] Examples — Matplotlib 3.5.1 documentation.
https://matplotlib.org/stable/gallery/index.html.

[9] Hiroshi Hashimoto and Koji Makino.
Python コンピュータシミュレーション入門.
オーム社, 2021.

[10] Kotaro Matsumoto.
早く知っておきたかった matplotlib の基礎知識、あるいは見た目の調整が捗る Artist の話.
https://qiita.com/skotaro/items/08dc0b8c5704c94eafb9.

[11] Makoto Matsumoto.
Mersenne Twister: A random number generator (since 1997/10).
http://www.math.sci.hiroshima-u.ac.jp/m-mat/MT/mt.html.

[12] Takahiro Ohmi.
C 言語による乱数生成.
https://omitakahiro.github.io/random/random_variables_generation.html.