

シミュレーション実習

中間レポート

南英幸

2022年5月10日

1 Buffon の針

1.1 解析的解答

y_s と θ の確率密度関数を、それぞれ、 $\Phi_1(y_s)$ と $\Phi_2(\theta)$ とすると、それらの確率密度関数は一様分布であるため、

$$\Phi_1(y_s) = \begin{cases} \frac{2}{d} : 0 \leq y_s \leq \frac{d}{2} \\ 0 : \text{otherwise} \end{cases} \quad (1)$$

$$\Phi_2(\theta) = \begin{cases} \frac{2}{\pi} : 0 \leq \theta \leq \frac{\pi}{2} \\ 0 : \text{otherwise} \end{cases} \quad (2)$$

となる。 $\Phi_1(y_s)$ と $\Phi_2(\theta)$ は互いに独立であるため、同時確率密度関数 $\Phi(y_s, \theta)$ は、単純に積を取ればよく、

$$\Phi(y_s, \theta) = \begin{cases} \frac{4}{\pi d} : 0 \leq y_s \leq \frac{d}{2}, 0 \leq \theta \leq \frac{\pi}{2} \\ 0 : \text{otherwise} \end{cases} \quad (3)$$

となる。今、 $l < d$ であるため、針が線と交わる条件は、 $\frac{l}{2} \cos \theta \geq y_s$ である。よってその確率は、

$$\int_{\theta=0}^{\theta=\frac{\pi}{2}} \int_{y_s=0}^{\frac{l}{2} \cos \theta} \Phi(y_s, \theta) dy_s d\theta = \frac{2l}{\pi d} \quad (4)$$

である。

1.2 (2) と (3) のシミュレーションプロット

シミュレーションに用いた、ソースコード (buffon.cpp) と、それをプロットするためのソースコード (buffon.py) を以下に示す。

ソースコード 1 buffon.cpp

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

```

2  #include <math.h>
3  #include <iostream>
4  #include <iomanip>
5  #include <fstream>
6  #include <time.h>
7  #include "MT.h"
8
9  double calc_buffon(int n, double d, double l);
10
11 int main(){
12     const double d = 2.0;
13     const double l = 1.0;
14     int n_lim = 100;
15     std::ofstream outfile;
16     std::string filename = "calc_buffone.csv";
17     outfile.open(filename);
18     outfile << "n_lim,pi,pi_error" << std::endl;
19     double answer, pi, pi_error;
20     while (n_lim <= (int)1e8)
21     {
22         answer = calc_buffon(n_lim, d, l);
23         pi = 1 / answer;
24         pi_error = fabs(pi - M_PI);
25         outfile << n_lim << "," << std::setprecision(7) << pi << "," << std::setprecision(7) <<
            pi_error << std::endl;
26         std::cout << n_lim << "," << std::setprecision(7) << pi << "," << std::setprecision(7) <<
            pi_error << std::endl;
27         n_lim *= 1.2;
28     }
29     n_lim = (int)1e8;
30     answer = calc_buffon(n_lim, d, l);
31     pi = (2.0 * l) / (answer * d);
32     pi_error = fabs(pi - M_PI);
33     outfile << n_lim << "," << std::setprecision(7) << pi << "," << std::setprecision(7) << pi_error
        << std::endl;
34     std::cout << n_lim << "," << std::setprecision(7) << pi << "," << std::setprecision(7) <<
        pi_error << std::endl;
35     outfile.close();
36     return 0;
37 }
38
39 double calc_buffon(int n, double d, double l){
40     int count = 0;
41     init_genrand(time(NULL));
42     for (int i = 0; i < n + 1; i++)
43     {
44         double y_s = d / 2 * genrand_res53();
45         double theta = genrand_res53() / 2 * M_PI;
46         if (1 / 2 * cos(theta) > y_s)

```

```

47     {
48         count++;
49     }
50 }
51 double p = count / (double)n;
52 return p;
53 }

```

ソースコード 2 buffon.py

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3 import pandas as pn
4 import math
5
6 plt.rcParams['text.usetex'] = True
7 xrange = (1e2, 1e8)
8 yrange_1 = (2.5, 4)
9 yrange_2 = (1e-6, 1)
10 xtitle = 'Number_of_trials'
11
12 fig = plt.figure(figsize=(8.27, 11.69))
13 plt.subplots_adjust(hspace=0.3)
14 csv_path = 'calc_buffone.csv'
15 csv_input = pn.read_csv(csv_path, header=0)
16 n = csv_input['n_lim'].to_numpy(dtype=int)
17 pi = csv_input['pi'].to_numpy(dtype=float)
18 pi_error = csv_input['pi_error'].to_numpy(dtype=float)
19
20 ax1 = fig.add_subplot(211, xscale='log', xlim=xrange, ylim=yrange_1)
21 ax1.plot(n, pi, 'x-', ms=5, color='r', label=r'Calculated_\pi$')
22 ax1.plot(xrange, [math.pi, math.pi], '--', color='k', label='$\pi$')
23 ax1.legend(loc=0, borderaxespad=0, fontsize=15)
24 ax1.set_title(r'Calculate_\pi$ by_buffon', fontsize=20)
25 ax1.set_xlabel(xtitle, fontsize=15)
26 ax1.set_ylabel('Calculated_data', fontsize=15)
27 ax1.spines['top'].set_linewidth(1.5)
28 ax1.spines['bottom'].set_linewidth(1.5)
29 ax1.spines['left'].set_linewidth(1.5)
30 ax1.spines['right'].set_linewidth(1.5)
31
32 ax2 = fig.add_subplot(212, xscale='log', yscale='log', ylim=yrange_2)
33 ax2.plot(n, pi_error, '+-', ms=5, color='b', label='error')
34 x = np.linspace(100, int(1e8), int(1e3))
35 y = 1 / x ** 0.5
36 ax2.plot(x, y, '--', color='k', label=r'$\propto\frac{1}{\sqrt{n}}$')
37 ax2.legend(loc=0, borderaxespad=0, fontsize=15)
38 ax2.set_title(r'Error_between_calculate_and_\pi$', fontsize=20)
39 ax2.set_xlabel(xtitle, fontsize=15)

```

```

40 ax2.set_ylabel('Error', fontsize=15)
41 ax2.spines['top'].set_linewidth(1.5)
42 ax2.spines['bottom'].set_linewidth(1.5)
43 ax2.spines['left'].set_linewidth(1.5)
44 ax2.spines['right'].set_linewidth(1.5)
45 plt.savefig('calc_buffon_plot.png')

```

また、シミュレーションした時の、円周率を計算した時のプロットと、理論値 π とのずれ量をを図 1 に示す。その際 $d = 2.0, l = 1.0$ とし、試行回数に対する計算結果と理論値との誤差について、参考として $1/\sqrt{n}$ を図示した。

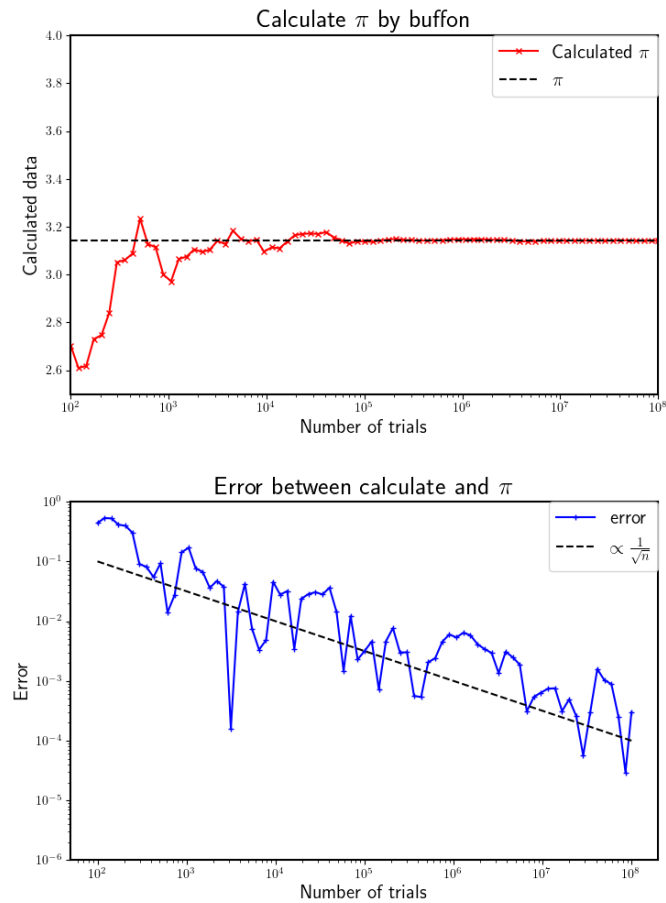


図 1

上図が試行回数に対する計算結果の分布、下図が理論値 π とのずれ量をプロットしたものである。

2 溶媒中におけるばねで繋がれた物体の運動

2.1 小球の運動の条件について

$x(t)$ の特殊解を $e^{\lambda t}$ とし、運動方程式に、代入して整理すると、

$$\lambda^2 + \frac{\zeta}{m}\lambda + \frac{k}{m} = 0 \quad (5)$$

となる。(5) の判別式を D とすると、 $D < 0$ の時減衰振動であり、 $D > 0$ の時過減衰となり、 $D = 0$ の時臨界減衰となる。よって、

$$\begin{cases} D < 0 \Leftrightarrow \left(\frac{\zeta}{m}\right)^2 < \frac{4k}{m} : \text{減衰振動} \\ D > 0 \Leftrightarrow \left(\frac{\zeta}{m}\right)^2 > \frac{4k}{m} : \text{過減衰} \\ D = 0 \Leftrightarrow \left(\frac{\zeta}{m}\right)^2 = \frac{4k}{m} : \text{臨界減衰} \end{cases} \quad (6)$$

である。

2.2 運動方程式の離散化

まず、問題文で与えられた運動方程式を、 $\frac{dv}{dt}, v(t), x(t)$ を用いて表すと、

$$m \frac{dv}{dt} = -\zeta v(t) - kv(t) \quad (7)$$

となり、これを離散化すると、

$$\begin{aligned} m \frac{v(t + \Delta t) - v(t)}{\Delta t} &= -\zeta v(t) - kv(t) \\ \Leftrightarrow v(t + \Delta t) &= \left(1 - \frac{\zeta}{m}\Delta t\right) v(t) - \frac{k}{m}x(t)\Delta t \end{aligned} \quad (8)$$

となる。さらに、無次元化するために、 $x = a\tilde{x}$, $t = t_0\tilde{t}$, $\dot{x} = v = (a/t_0)\tilde{v}$ とすると $\tilde{v}(\tilde{t})$ の二項間漸化式は、

$$\begin{aligned} \frac{a}{t_0}\tilde{v}(\tilde{t} + \Delta\tilde{t}) &= \left(1 - \frac{\zeta}{m}t_0\Delta\tilde{t}\right) \frac{a}{t_0}\tilde{v}(\tilde{t}) - \frac{k}{m}a\tilde{x}(\tilde{t})t_0\Delta\tilde{t} \\ \Leftrightarrow \tilde{v}(\tilde{t} + \Delta\tilde{t}) &= \left(1 - \frac{\zeta}{m}t_0\Delta\tilde{t}\right) \tilde{v}(\tilde{t}) - \frac{k}{m}t_0^2\tilde{x}(\tilde{t})\Delta\tilde{t} \end{aligned} \quad (9)$$

となる。また、 $x(t)$ に関しては、

$$x(t + \Delta t) = x(t) + v(t + \Delta t)\Delta t \quad (10)$$

$$= x(t) + \left(1 - \frac{\zeta}{m}\Delta t\right) v(t)\Delta t - \frac{k}{m}x(t)\Delta t^2 \quad (11)$$

であるため、 v と同じように無次元化すると、

$$\tilde{x}(\tilde{t} + \Delta\tilde{t}) = \tilde{x}(\tilde{t}) + \tilde{v}(\tilde{t} + \Delta\tilde{t})\Delta\tilde{t} \quad (12)$$

$$= \left(1 - \frac{\zeta}{m}t_0\Delta\tilde{t}\right)\tilde{v}(\tilde{t})\Delta\tilde{t} + \left(1 - \frac{k}{m}t_0^2\right)\tilde{x}(\tilde{t})\Delta\tilde{t}^2 \quad (13)$$

となる。

2.3 パラメータの導入

$t_d = \frac{m}{\zeta}$, $t_s = \sqrt{\frac{m}{k}}$ を用いて、式 (9),(13) を書き換えると、

$$\tilde{v}(\tilde{t} + \Delta\tilde{t}) = \left(1 - \frac{t_0}{t_d}\Delta\tilde{t}\right)\tilde{v}(\tilde{t}) - \left(\frac{t_0}{t_s}\right)^2\tilde{x}(\tilde{t})\Delta\tilde{t} \quad (14)$$

$$\tilde{x}(\tilde{t} + \Delta\tilde{t}) = \tilde{x}(\tilde{t}) + \left(1 - \frac{t_0}{t_d}\Delta\tilde{t}\right)\tilde{v}(\tilde{t})\Delta\tilde{t} - \left(1 - \frac{t_0^2}{t_s^2}\right)\tilde{x}(\tilde{t})\Delta\tilde{t}^2 \quad (15)$$

となる。

2.4 シミュレーション結果

以上のことを踏まえて、物体の抵抗がある運動方程式について数值的に解析した。そのソースコード (dampingEoM.cpp) とその結果をプロットするソースコード (dampingEoM.py) を以下に示す。ただし、 $a = 10$ とし、振動のパラメータは減衰振動、過減衰、臨界減衰の3つの条件が出るように t_d の値を定めた。

ソースコード 3 dampingEoM.cpp

```

1 #include <stdio.h>
2 #include <math.h>
3 #include <iostream>
4 #include <iomanip>
5 #include <fstream>
6 #include <cmath>
7
8 void damping(double a, double dt, double t_d, double t_0);
9
10 int main(){
11     const double a = 10.;
12     double dt = 1e-3, t_d = 0.25, t_0 = 1.0;
13     damping(a, dt, t_d, t_0);
14     t_d = 0.1;
15     damping(a, dt, t_d, t_0);
16     t_d = 4;
17     damping(a, dt, t_d, t_0);
18     return 0;
19 }
20
21 void damping(double a, double dt, double t_d, double t_0){

```

```

22     char filename[128];
23     std::ofstream file;
24     int i = 0, div = (int)1/dt / 10;
25     double v = a / t_0, x = 0.;
26     sprintf(filename,"pos_%.2f_%.4f_%.2f_%.2f.csv", a, dt, t_d, t_0);
27     file.open(filename);
28     file << "t,v,x" << std::endl;
29     file << (double)i*dt << "," << v << "," << x << std::endl;
30     std::cout << (double)i*dt << "," << v << "," << x << std::endl;
31     while(fabs(v) > 1e-4){
32         v = v - (t_0/t_d)*dt*v - x*dt;
33         x = x + v*dt;
34         i++;
35         if(i % div == 0){
36             file << (double)i*dt << "," << v << "," << x << std::endl;
37             std::cout << (double)i*dt << "," << v << "," << x << std::endl;
38         }
39     }
40     file.close();
41 }

```

ソースコード 4 dampingEoM.py

```

1 import matplotlib.pyplot as plt
2 import pandas as pn
3 import matplotlib.cm as cm
4
5
6 def draw(pos, a, dt, t_d, t_0):
7     csv_path = 'pos_{:.2f}_{:.4f}_{:.2f}_{:.2f}.csv'.format(a, dt, t_d, t_0)
8     position = int('22{:.2f}'.format(pos))
9     plt.subplot(position)
10    csv_input = pn.read_csv(csv_path, header=0)
11    t = csv_input['t'].to_numpy(dtype=float)
12    v = csv_input['v'].to_numpy(dtype=float)
13    x = csv_input['x'].to_numpy(dtype=float)
14    plt.plot(t, x, 'x-', ms=3, color=cm.jet(t_d), label=r'$t_s$={:.2f}, $t_0$={:.2f}'.format(t_d,
        t_0))
15    plt.legend(loc=0, borderaxespad=0, fontsize=15)
16    plt.xlabel(r'$t/t_0(=\tilde{t})$', fontsize=15)
17    plt.ylabel(r'$x(t)/a(=\tilde{x}(t))$', fontsize=15)
18    if t_d < t_0 / 4:
19        plt.title('over_damping', fontsize=20)
20    elif t_d > t_0 / 4:
21        plt.title('under_damping', fontsize=20)
22    else:
23        plt.title('critical_damping', fontsize=20)
24
25

```

```

26 plt.rcParams['text.usetex'] = True
27 a = 10.
28 fig = plt.figure(figsize=(11.69, 8.27), tight_layout=True)
29 fig.suptitle('simulated_position(a={:.2f})'.format(a), fontsize=30)
30 dt = 1e-3
31 t_d = 0.1
32 t_0 = 1.
33 draw(1, a, dt, t_d, t_0)
34 t_d = 0.25
35 draw(2, a, dt, t_d, t_0)
36 t_d = 4.
37 draw(3, a, dt, t_d, t_0)
38
39 plt.show()
40 plt.savefig('plot_dampingEoM.png')

```

また、シミュレーション結果をプロットしたものを図2に示す。

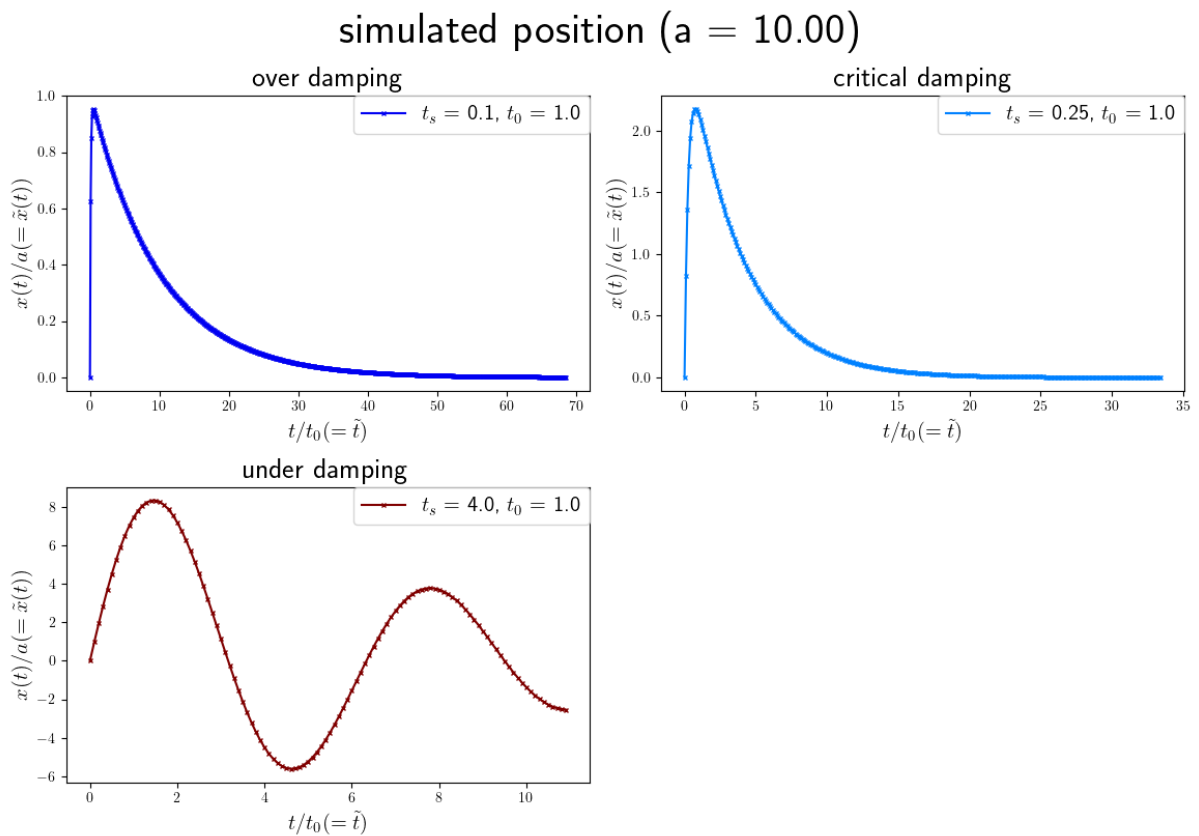


図2
それぞれの t_d に対する、位置の時間発展