1_buffon

2024 年 5 月 27 日

1　シミュレーション実習中間レポート課題の回答

[ ]:
```
# Google Colab setup
!pip install numpy==1.22.0
!pip install -U polars seaborn
!pip install sympy
```

[4]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import polars as pl
```

1.1　第 1 問

1.1.1　1-1

[ ]:
```
# define parameterss
d = 8
l_list = [1, 2, 6, 8]

# Generate thera array to plot on graph
theta_array = np.arange(0.01, np.pi/2, np.pi/256)
# Generate y_c array to plot on graph
y_c_array = np.arange(0.01, d/2, d/256)

def condition(l, y_c, theta):
    cond = l * np.cos(theta) / (2 * y_c) > 1
    return cond
```

```
def touched_patterns(l, y_c_array, theta_array):
    patterns = []
    for y_c in y_c_array:
        for theta in theta_array:
            if condition(l, y_c, theta):
                patterns.append([l, y_c, theta])
    return patterns

patterns_dict = {l: [] for l in l_list}
print(isinstance(patterns_dict, dict))
```

```
True
```

[ ]:
```
# Simulate bar-touch and generate DataFrame of touched paramters patterns
df = pl.DataFrame()
for l in patterns_dict.keys():
    patterns_dict[l] = touched_patterns(l, y_c_array, theta_array)

    df_temp = pl.DataFrame(patterns_dict[l])
    df = df.vstack(df_temp)
df.columns = ['l', 'y_c', 'theta']
print(df.shape)
print(df.head(5))
```

1.1.2 1-1 結果

■シミュレーションの概要

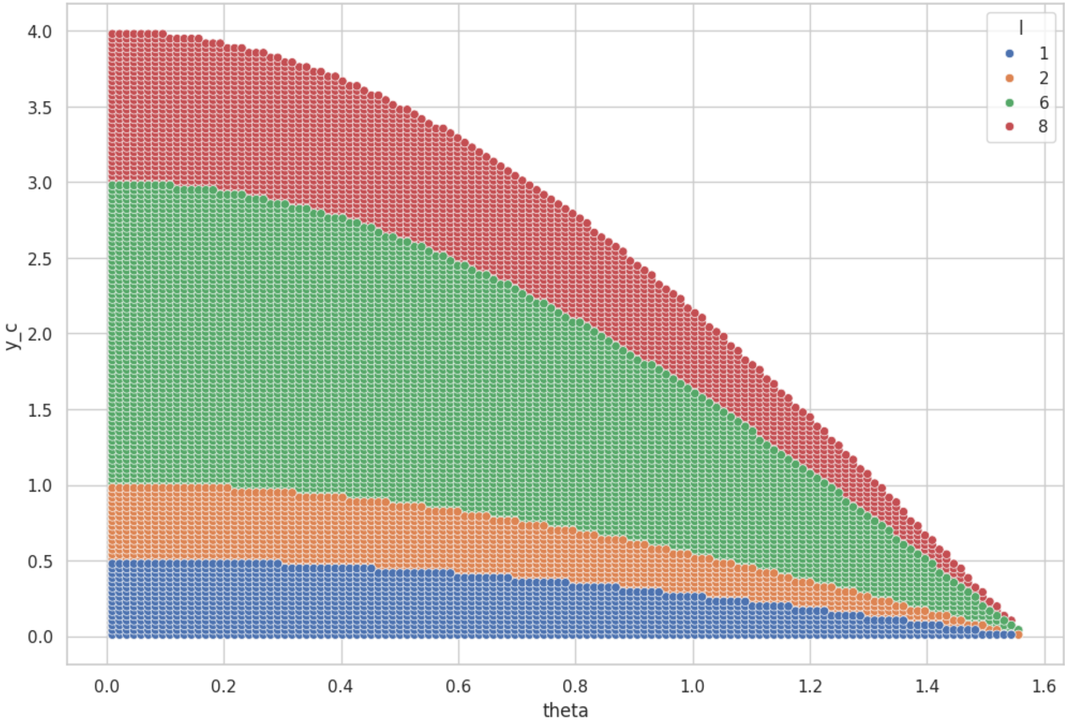
1. 以下は $d = 8$ としたときの針が棒と交わる $\theta - y_c$ の条件を数値計算した結果である。
2. 針の長さを表す l を $[1, 2, 6, 8]$ の 4 つの場合でシミュレーションした。

■結果

1. グラフの通り l が長くなればなるほど針が棒と交わる θ, y_c の条件が緩くなることがわかる。

[ ]:
```
# Sort by l for displaying
df = df.sort('l', descending=True)
# Plot
sns.set_theme(style="whitegrid")
```

```python
plt.figure(figsize=(12, 8))
sns.scatterplot(data=df, x='theta', y='y_c', hue='l', palette='deep')
plt.show()
```



1.2 1-2

- 棒を落とした時に取りうる -y_c 平面上の全領域の面積を s_all
- 1-1 で得られた棒と針が交わる場合の領域の面積を s_touched とする。p(l) = s_all / s_touched とすると p(l) が棒と針が交わる確率を表せる。

$S_{\text{all}} = d/2 \times \pi/2 = d \times \pi/4$

S_{touched} は $y_c \leq l/2 \times \cos\theta$ を満たす領域の面積。以下のコードブロックで計算する。

[ ]:
```python
from sympy import *


x = Symbol("x")
l = Symbol("l")
theta = Symbol("theta")
y_c = Symbol("y_c")
```

```
s_touched = integrate(l / 2 * cos(theta), (theta, 0, pi/2))
print(s_touched)
```

```
l/2
```

1.2.1 1-2 結果

以上の計算より $S_{\text{touched}} = l/2$

また $S_{\text{all}} = d\pi/4$ であるから求めたい確率は確かに $p(l) = \frac{2l}{\pi d}$ となる。

1.2.2 1-3

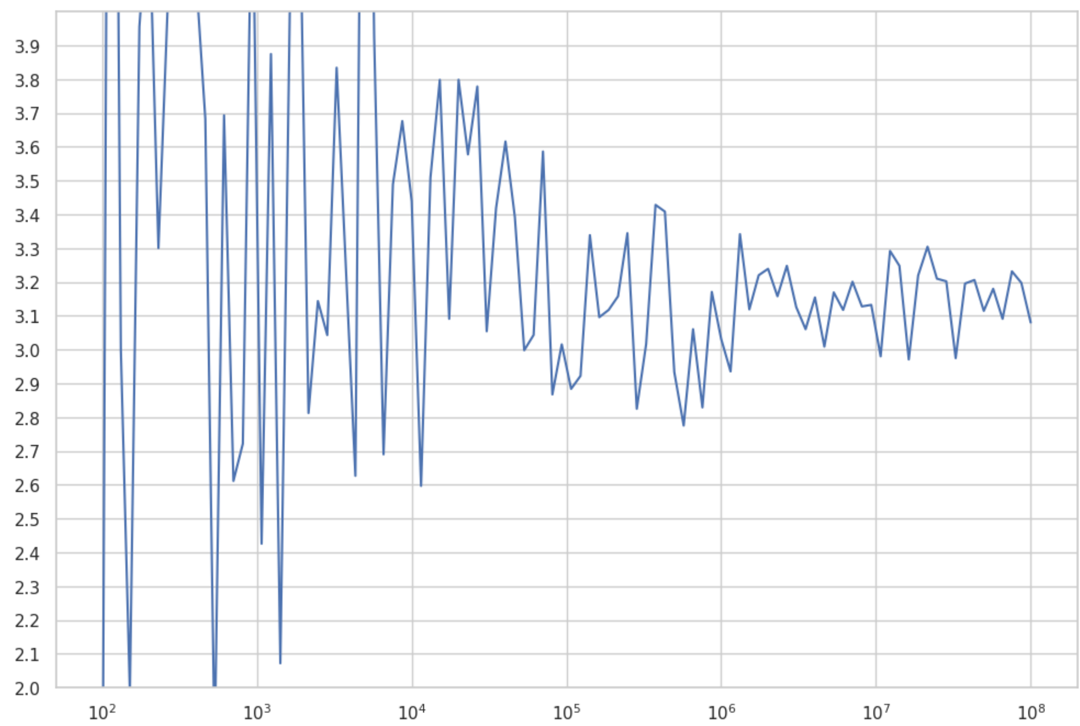
1-2 で求めた確率 p(l) / l = 1/ (pi * d) となるので y_c, theta の一様乱数を生成して数値的に p(l) の近似値を得れば 1/ (pi *d) すなわり pi の近似値を得ることができる。

[ ]:
```
# Google Colab setup
!nvidia-smi
!pip uninstall -y cupy-cuda12x
!pip install cupy-cuda11x
```

[2]:
```
import cupy as cp
import numpy as np
```

[6]:
```
# 高速化のため qupy で GPGPU 行列演算させる
def condition(l, y_c, theta):
    return l * cp.cos(theta) / (2 * y_c) > 1

def pi_approxer(d: int, l: int, count_pairs: int = 10**2):
    count_pairs_sqrt = int(np.sqrt(count_pairs))
    theta_array = np.random.uniform(0, np.pi / 2, count_pairs_sqrt).astype(np.
  ↪float16)
    y_c_array = np.random.uniform(0, d / 2, count_pairs_sqrt).astype(np.float16)

    theta_array = cp.asarray(theta_array, dtype=cp.float16)
    y_c_array = cp.asarray(y_c_array, dtype=cp.float16)

    # Use numpy broadcasting to calculate the condition for all pairs
    touched = condition(l, y_c_array[:, np.newaxis], theta_array)
```

```
    # Count the number of True values
    count_touched = np.sum(touched)
    count_all_pairs = count_pairs

    posibility = count_touched / count_all_pairs
    pi = 2 * l / d / posibility
    return pi

count_throw_array = np.logspace(2, 8, 100).astype(int)
pi_array = np.array([pi_approxer(8, 2, count_throw).get() for count_throw in count_throw_array])

sns.set_theme(style="whitegrid")
plt.figure(figsize=(12, 8))
sns.lineplot(x=count_throw_array, y=pi_array)
plt.xscale('log')
plt.yticks(np.arange(2, 4, 0.1))
plt.ylim(2, 4)
# plt.show()
plt.savefig('pi_approximation.png')
```



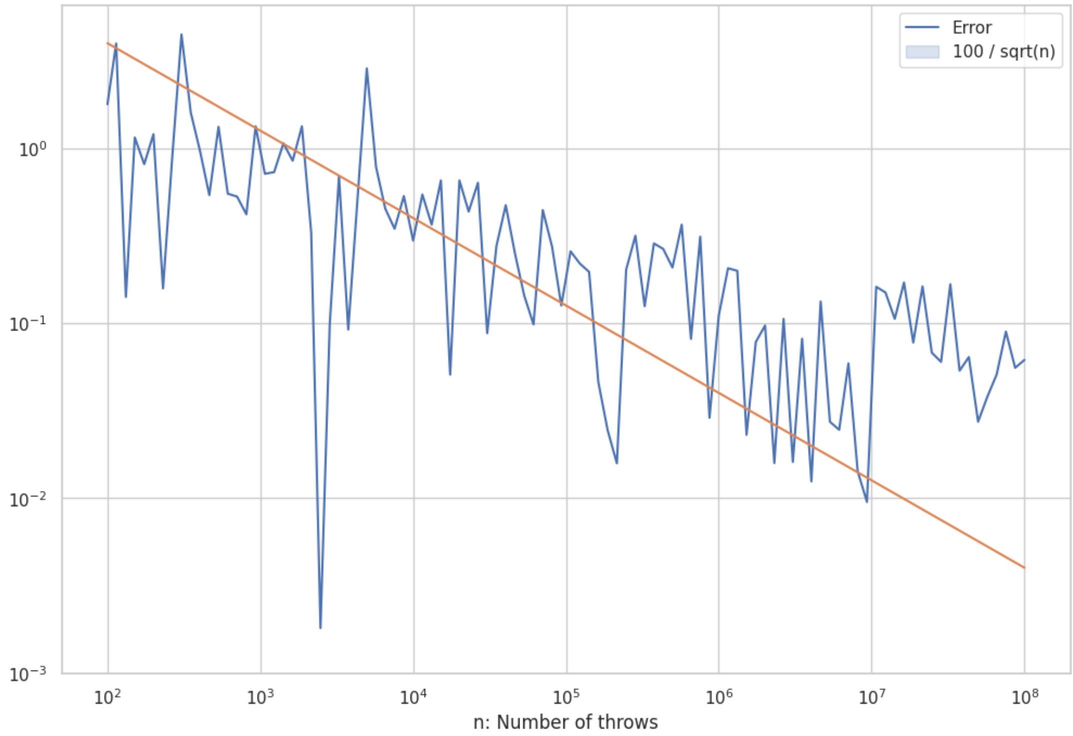
1.2.3 1-3

以下の画像が 10**2 から 10**8 回針を落としたシミュレーションで得られた pi の近似値の推移である。

収束速度は遅いが 3.14159... に収束する様子がわかる。

1.2.4 1-4

[7]:
```python
# 試行回数 n と 誤差 error の関係
c_over_root_n = 40 / np.sqrt(count_throw_array)

error_array = np.abs(np.pi - pi_array)
sns.set_theme(style="whitegrid")
plt.figure(figsize=(12, 8))
sns.lineplot(x=count_throw_array, y=error_array)
sns.lineplot(x=count_throw_array, y=c_over_root_n)
plt.xscale('log')
plt.yscale('log')
plt.xlabel('n: Number of throws')
plt.yticks(np.logspace(-3, 0, 4))
plt.legend(['Error', '100 / sqrt(n)'])
# plt.ylim(1e-3, 10**2)
# plt.show()
plt.savefig('error_approximation.png')
```



1.2.5　1-4 結果

- 上で行った数値計算の結果 Buffon の針による円周率の近似は誤差収束速度が $\sqrt{N}$ より遅いことがわかった。
- 非効率な円周率計算アルゴリズムと言える。

2_simple_harmonic_motion

2024 年 5 月 27 日

0.1　第２問

一端を固定した，ばね定数 k のばねに，質点とみなせる質量 m の小球をつけ，滑らかな水平面上を 1 次元運動させる．ここで小球に働く抵抗力や小球に働く熱揺動力は無視できるものとする．この時，適当な座標系を敷き，時刻 t における小球の位置を $x(t)$ とすれば，物体の運動方程式は，$m\ddot{x}(t) = -kx(t)$ と表され，物体が周期運動（単振動）する解析解を得ることができる．いま，これを数値計算を用いて解くことにしよう．以下の各問に答えよ．

0.1.1　2-1

$\tilde{x}$, $\tilde{t}$, $\tilde{v}$ を長さ, 時間, 速度に関する無次元の変数であるとし, 問題にて与えた実際の物理量との関係は $x = a\tilde{x}$, $t = t_0\tilde{t}$, $\dot{x} = v = (a/t_0)\tilde{v}$ であるとする．ここで $a[m]$, $t_0[s]$ は適当に定めた長さと時間である．上記の関係を用いることで運動方程式を無次元であることを示せ。

実際の物理量と無次元変数の関係は以下の通り。

$$x = a\tilde{x}, \quad t = t_0\tilde{t}, \quad \dot{x} = v = \frac{a}{t_0}\tilde{v}$$

これらの関係を用いて運動方程式を書き表す。

$$m\ddot{x}(t) = -kx(t)$$

左辺の $\ddot{x}(t)$ は 2 階微分なので無次元化すると

$$\ddot{x} = \frac{d^2x}{dt^2} = \frac{d}{dt}\left(\frac{dx}{dt}\right) = \frac{d}{dt}\left(\frac{a\tilde{v}}{t_0}\right) = \frac{a}{t_0}\frac{d\tilde{v}}{dt}$$

さらに時間の無次元化を使い $\frac{d}{dt}$ を $\frac{d}{d\tilde{t}}$ に置き換える

$$\frac{d\tilde{v}}{dt} = \frac{d\tilde{v}}{d\tilde{t}} \cdot \frac{d\tilde{t}}{dt} = \frac{d\tilde{v}}{d\tilde{t}} \cdot \frac{1}{t_0}$$

したがって、2 階微分は以下のようになる

$$\ddot{x} = \frac{a}{t_0}\frac{d}{d\tilde{t}}\left(\frac{d\tilde{x}}{d\tilde{t}}\right) = \frac{a}{t_0} \cdot \frac{1}{t_0}\frac{d^2\tilde{x}}{d\tilde{t}^2} = \frac{a}{t_0^2}\frac{d^2\tilde{x}}{d\tilde{t}^2}$$

左辺を整理すると

$$m\ddot{x}(t) = m\frac{a}{t_0^2}\frac{d^2\tilde{x}}{d\tilde{t}^2}$$

右辺の $ x(t) $ を無次元変数で置き換える

$$-kx(t) = -ka\tilde{x}$$

運動方程式は次のようになる

$$m\frac{a}{t_0^2}\frac{d^2\tilde{x}}{d\tilde{t}^2} = -ka\tilde{x}$$

$ a $ を両辺から除去すると

$$m\frac{1}{t_0^2}\frac{d^2\tilde{x}}{d\tilde{t}^2} = -k\tilde{x}$$

最終的に無次元化された運動方程式は以下の通り。

$$\ddot{\tilde{x}} = -\left(\frac{kt_0^2}{m}\right)\tilde{x}$$

0.1.2　2-2

ここで、$t_0 = \sqrt{m/k}$ とおくと単振動の運動方程式は

$$\ddot{\tilde{x}} = -\tilde{x}$$

となる。

0.1.3　2-3

■陽的 Euler 法による漸化式　速度の漸化式

$$\tilde{v}(t + \Delta\tilde{t}) = \tilde{v}(t) - \Delta\tilde{t} \cdot \tilde{x}(t)$$

位置の漸化式

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

0.1.4　2-4

半陰的 Euler 法による漸化式は次のようになる。

速度の漸化式

$$\tilde{v}(t+\Delta\tilde{t}) = \tilde{v}(t) - \Delta\tilde{t}\cdot\tilde{x}(t)$$

位置の漸化式

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

速度の更新は同じだが、位置の更新に今のステップの速度を使うのが陽的 Euler 法なのに対して、半陰的 Euler 法では位置の更新に次の時間ステップでの速度を使う。

0.1.5 2-5

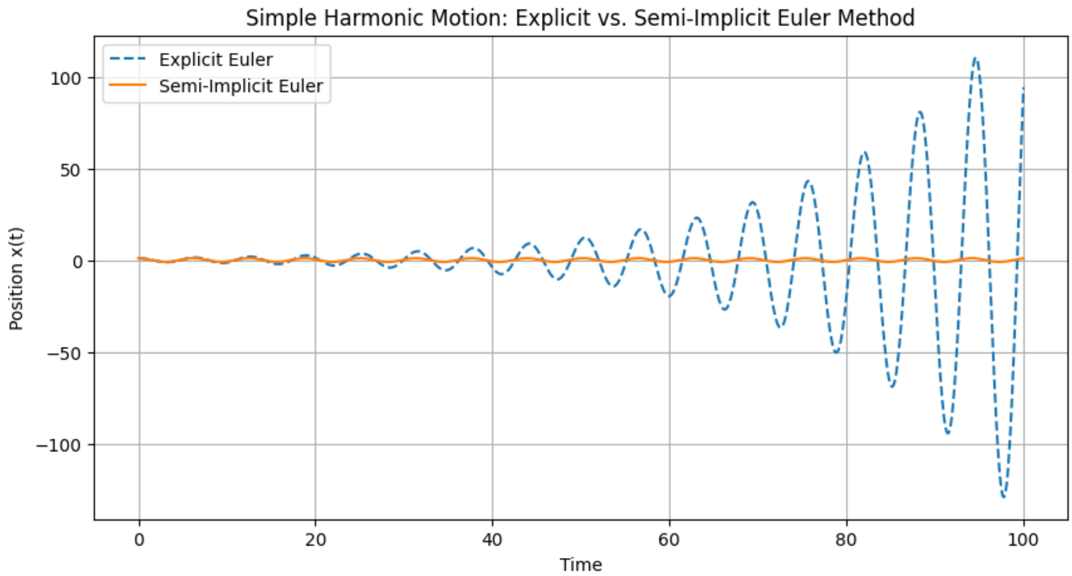
[2]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[6]:
```
# Set physical parameters
a = 1.0  # initial position
v0 = 0.0  # initial velocity
dt = 0.1  # time step
t_max = 100  # max time
num_steps = int(t_max / dt)  # steps
```

[7]:
```
# 陽的 Euler 法
x_explicit = np.zeros(num_steps + 1)
v_explicit = np.zeros(num_steps + 1)
x_explicit[0] = a
v_explicit[0] = v0

for i in range(num_steps):
    v_explicit[i + 1] = v_explicit[i] - dt * x_explicit[i]
    x_explicit[i + 1] = x_explicit[i] + dt * v_explicit[i]

# 半陰的 Euler 法
x_semi_implicit = np.zeros(num_steps + 1)
v_semi_implicit = np.zeros(num_steps + 1)
x_semi_implicit[0] = a
v_semi_implicit[0] = v0

for i in range(num_steps):
    v_semi_implicit[i + 1] = v_semi_implicit[i] - dt * x_semi_implicit[i]
```

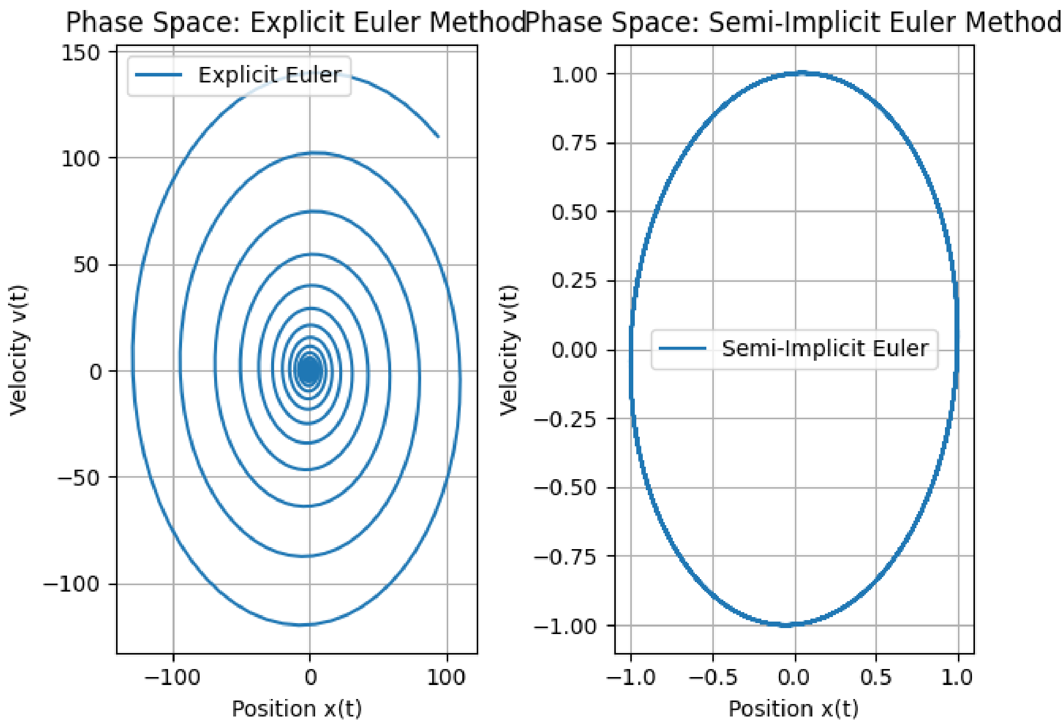
```
    x_semi_implicit[i + 1] = x_semi_implicit[i] + dt * v_semi_implicit[i + 1]
```

[8]:
```
# time steps
time = np.arange(0, t_max + dt, dt)

# Plot results
plt.figure(figsize=(10, 5))
plt.plot(time, x_explicit, label='Explicit Euler', linestyle='--')
plt.plot(time, x_semi_implicit, label='Semi-Implicit Euler', linestyle='-')
plt.xlabel('Time')
plt.ylabel('Position x(t)')
plt.title('Simple Harmonic Motion: Explicit vs. Semi-Implicit Euler Method')
plt.legend()
plt.grid()
plt.show()
```



[9]:
```
# 陽的 Euler 法の位相空間プロット
plt.subplot(1, 2, 1)
plt.plot(x_explicit, v_explicit, label='Explicit Euler')
plt.xlabel('Position x(t)')
plt.ylabel('Velocity v(t)')
plt.title('Phase Space: Explicit Euler Method')
```

```
plt.legend()
plt.grid()

# 半陰的 Euler 法の位相空間プロット
plt.subplot(1, 2, 2)
plt.plot(x_semi_implicit, v_semi_implicit, label='Semi-Implicit Euler')
plt.xlabel('Position x(t)')
plt.ylabel('Velocity v(t)')
plt.title('Phase Space: Semi-Implicit Euler Method')
plt.legend()
plt.grid()

plt.tight_layout()
plt.show()
```



3_central_limiting_theorem

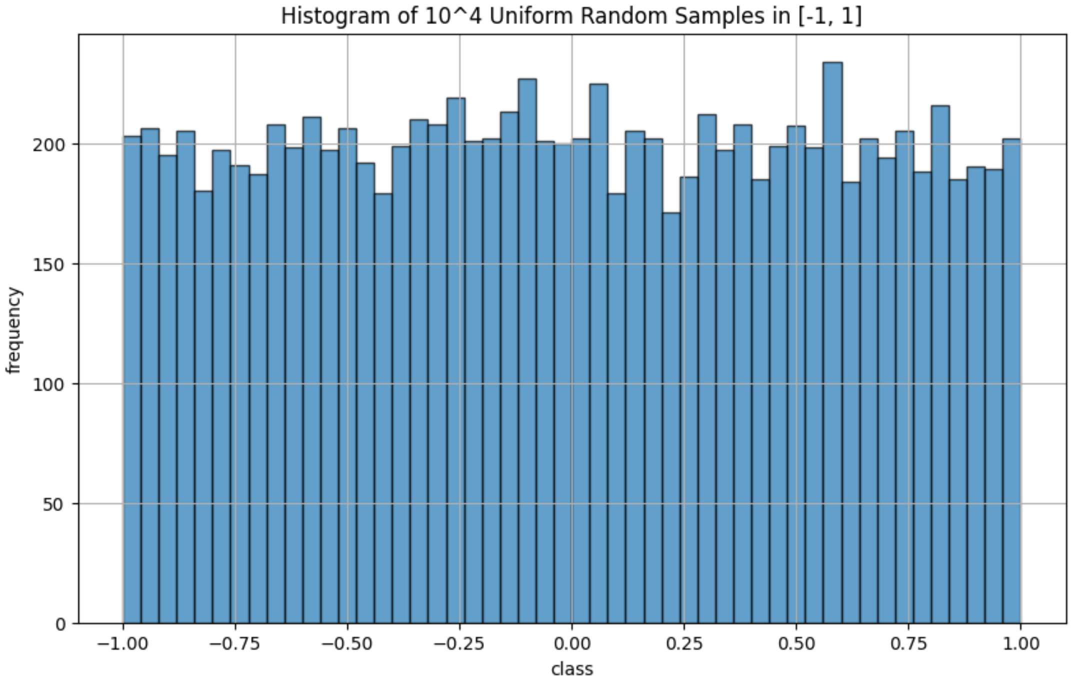
2024 年 5 月 27 日

0.1　第３問

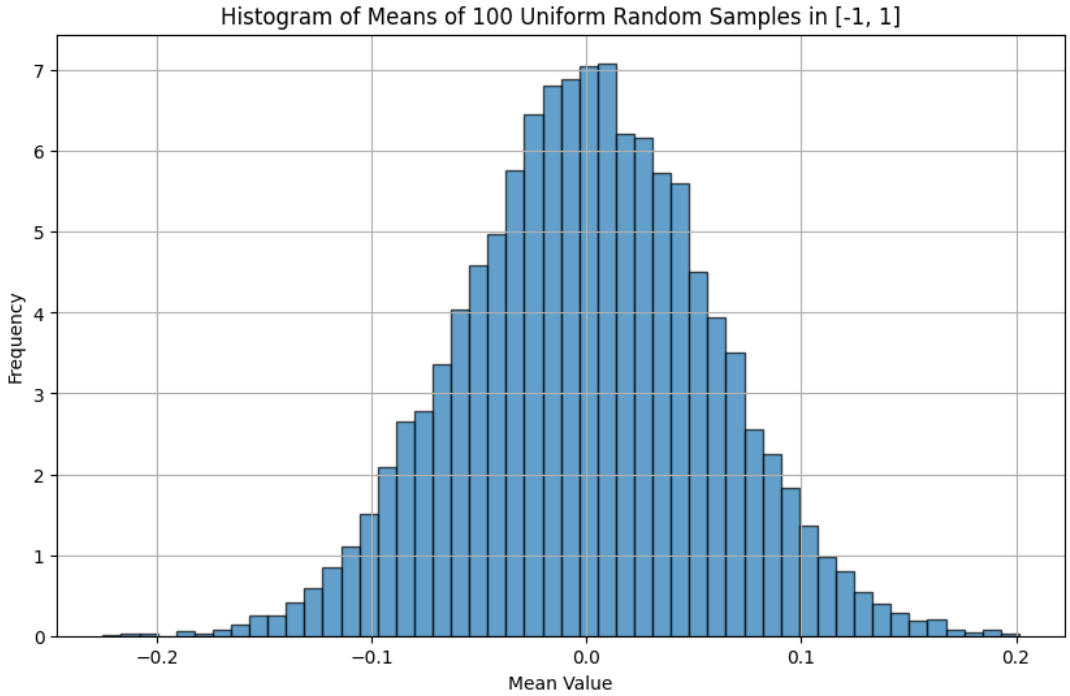
0.1.1　3-1

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[5]:
```
# 一様乱数を発生させる
num_samples = 10**4
uniform_random_samples = np.random.uniform(-1, 1, num_samples)

# 度数分布をプロットする
plt.figure(figsize=(10, 6))
plt.hist(uniform_random_samples, bins=50, edgecolor='black', alpha=0.7)
plt.xlabel('class')
plt.ylabel('frequency')
plt.title('Histogram of 10^4 Uniform Random Samples in [-1, 1]')
plt.grid(True)
plt.show()
```



[7]:
```
# Generate 10^6 uniform random samples
num_samples = 10**6
uniform_random_samples = np.random.uniform(-1, 1, num_samples)

# Compute the averages of 100 samples
sample_size = 100
num_averages = num_samples // sample_size
averages = np.mean(uniform_random_samples.reshape(num_averages, sample_size),␣
 ↪axis=1)

# Plot the histogram of the averages
plt.figure(figsize=(10, 6))
plt.hist(averages, bins=50, edgecolor='black', alpha=0.7, density=True)
plt.xlabel('Mean Value')
plt.ylabel('Frequency')
plt.title('Histogram of Means of 100 Uniform Random Samples in [-1, 1]')
plt.grid(True)
plt.show()
```



■解析的な考察　X_i を ([-1,1]) の範囲での一様乱数とすると、一様分布の平均 μ と分散 σ^2 は次の通りとなる。一様分布の平均：$\mu = \frac{a+b}{2} = \frac{-1+1}{2} = 0$

一様分布の分散：$\sigma^2 = \frac{(b-a)^2}{12} = \frac{(1-(-1))^2}{12} = \frac{4}{12} = \frac{1}{3}$

今 n 個の独立な一様乱数 X_i を考える。その平均を $\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$ とする。

サンプリング数 n とその平均値の分布の一般的性質を調べる。

まず $\bar{X}$ の平均と分散を求める。

平均 $\mathbb{E}[\bar{X}] = \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n} X_i\right] = \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}[X_i] = \frac{1}{n}\cdot n \cdot \mu = \mu = 0$

分散 $\mathrm{Var}(\bar{X}) = \mathrm{Var}\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right) = \frac{1}{n^2}\sum_{i=1}^{n}\mathrm{Var}(X_i) = \frac{1}{n^2}\cdot n \cdot \sigma^2 = \frac{\sigma^2}{n} = \frac{1/3}{n} = \frac{1}{3n}$

$Z = \frac{\bar{X}-\mathbb{E}[\bar{X}]}{\sqrt{\mathrm{Var}(\bar{X})}}$ を標準化すると、Z は平均 0、分散 1 の標準正規分布に従う。

Z を展開すると $Z = \frac{\bar{X}-0}{\sqrt{1/(3n)}} = \bar{X}\sqrt{3n}$

以上より n 個の独立な乱数の平均は、n が大きくなるにつれて正規分布に収束する。具体的には、平均 0、分散 $\frac{1}{3n}$ の正規分布に近づく。