マルチエージェントシステム藤井先生レポート 1

03-230966 河田顕帆

2024 年 1 月 31 日

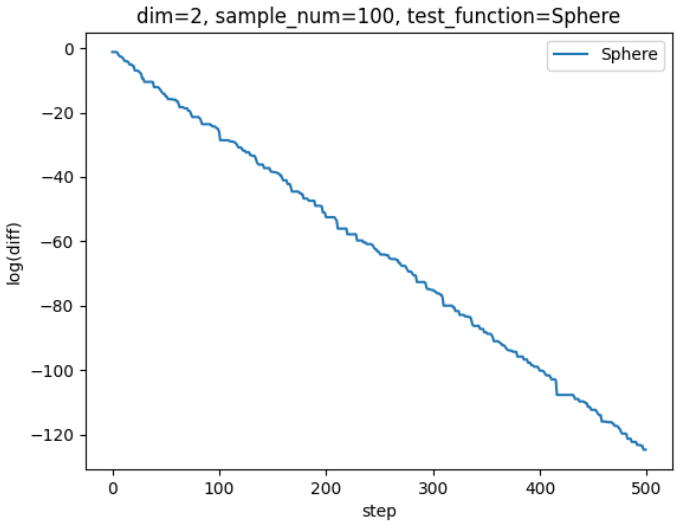
1　課題 1

私は、PSO のプログラムを作成して、6 つの関数に対して PSO の性能を評価した。以下のサンプル数は幅 10 に対して 100 サンプル取るようにしている。また、次元数は 2 と 20 の 2 種類を試した。step 数はいずれも 500 ステップである。
また、最適解との差は、2 乗ノルムで計算した。

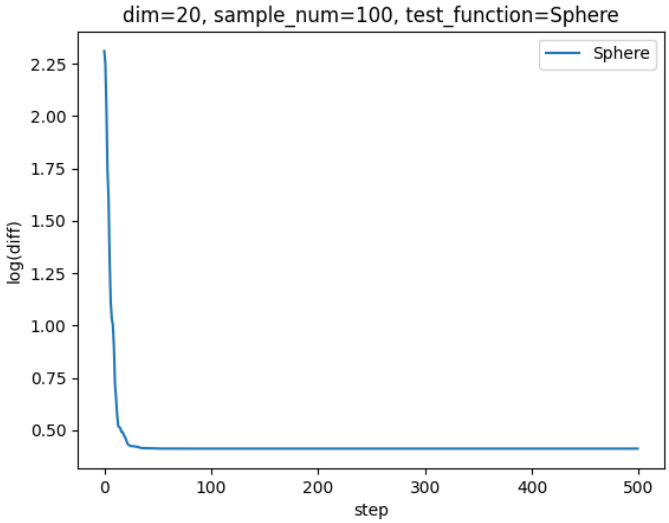
1. Sphere 関数
   関数の係数等は以下の設定で行った。

$$f(x) = \sum_{i=1}^{n} x_i^2, (-5.00 \leq x_i \leq 5.00) \tag{1}$$

   この場合、最小値は全次元 0 の x を与えた際の 0 である。
   以下に、次元数とサンプル数を変化させた際の結果を示す。グラフは、各ステップにおける最適解との差の対数をプロットしたものである。
   - 次元数: 2, サンプル数: 100, ステップ数: 500
     図から分かるようにステップを経るにつれて最適解との差の対数が単調にほぼ一様に小さくなっている。収束した様子は見られず、ステップ数を増加させれば収束した可能性がある。また、非常に最適値と近い値をとっていることがわかる。



- 次元数: 20, サンプル数: 100, ステップ数: 500
  図から分かるようにステップを経るにつれて最適解との差の対数が小さくなっているが、次元数が 2 の場合のようにほぼ一様に減少するのではなく、50step 程度までで急激に減少し、その後の減少は緩やかである。ここで収束したものと思われる。



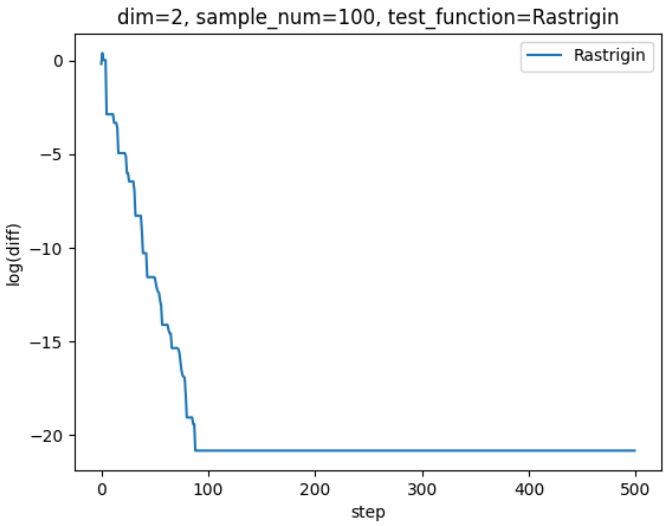
2. Rastrigin 関数関数の係数等は以下の設定で行った。

$$f(x) = 10n + \sum_{i=1}^{n}(x_i^2 - 10\cos(2\pi x_i)), (-5.00 \leq x_i \leq 5.00) \tag{2}$$

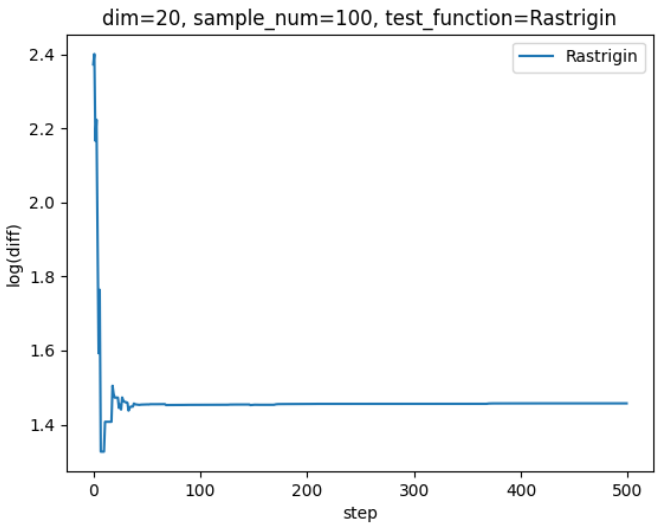
この場合、最小値は全次元 0 の x を与えた際の 0 である。
以下に、次元数とサンプル数を変化させた際の結果を示す。グラフは、各ステップにおける最適解との

差の対数をプロットしたものである。

- 次元数: 2, サンプル数: 100, ステップ数: 500
  図から分かるようにステップを経るにつれて最適解との差が 100 ステップまでは単調に小さくなっているものの、そこで収束したと思われる。最適値との誤差は非常に小さな点で収束したことがわかる。



- 次元数: 20, サンプル数: 100, ステップ数: 500
  一度急速に最適値へと近づいたものの、再び誤差が大きくなってから収束したと思われる。各ステップでの変動幅が大きかったせいで最適値を通り越してしまって局所最小値に陥ったと思われる。



3. Rosenbrock 関数

関数の係数等は以下の設定で行った。

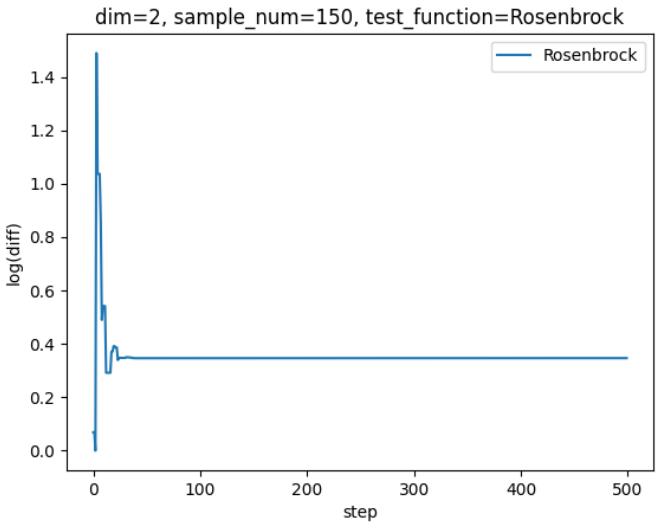
$$f(x) = \sum_{i=1}^{n-1}(100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2), (-5.00 \leq x_i \leq 10.00) \tag{3}$$

この場合、最小値は全次元 0 の x を与えた際の 0 である。

以下に、次元数とサンプル数を変化させた際の結果を示す。グラフは、各ステップにおける最適解との差の対数をプロットしたものである。

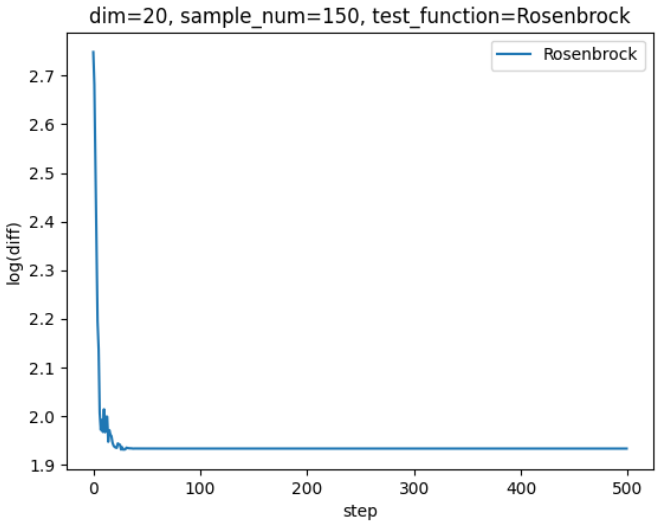
- 次元数: 2, サンプル数: 150, ステップ数: 500

  この場合も、一度急速に最適値へと近づいたものの、再び誤差が大きくなってから収束したと思われる。各ステップでの変動幅が大きかったせいで最適値を通り越してしまって局所最小値に陥ったと思われる。



- 次元数: 20, サンプル数: 150, ステップ数: 500

  数十ステップで収束したと思われる。誤差の大きさからすると、誤差が小さいわけではないので、局所最小値に陥った可能性がある。



4. Griewank 関数

   関数の係数等は以下の設定で行った。

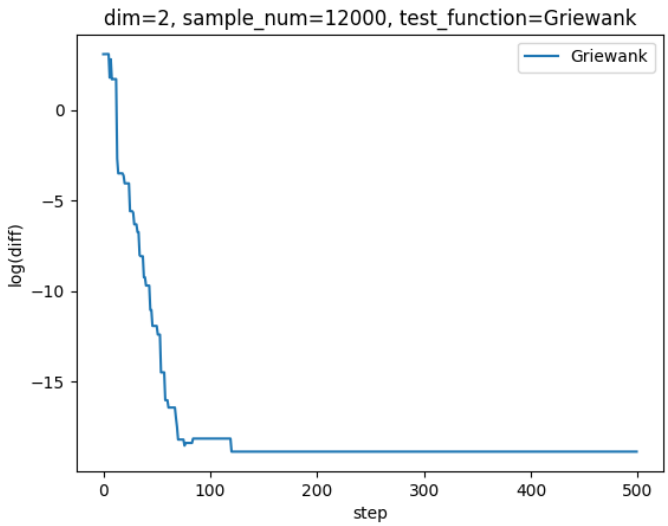
$$f(x) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}}) + 1, (-600.00 \leq x_i \leq 600.00) \tag{4}$$

   この場合、最小値は全次元 0 の x を与えた際の 0 である。

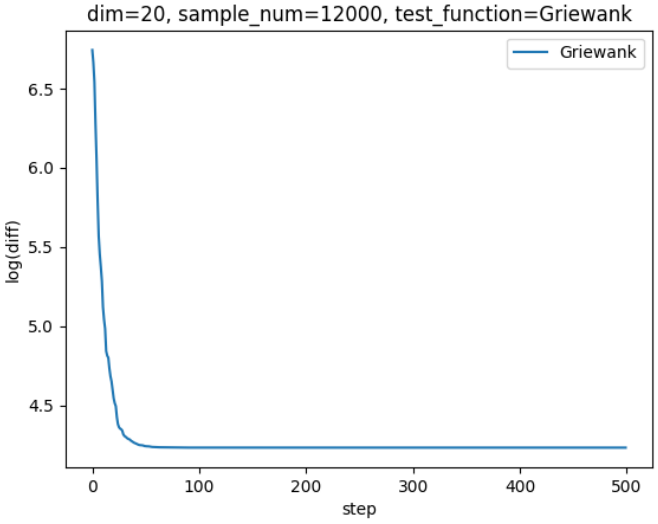
   以下に、次元数とサンプル数を変化させた際の結果を示す。グラフは、各ステップにおける最適解との差の対数をプロットしたものである。

   - 次元数: 2, サンプル数: 12000, ステップ数: 500

     100 ステップを過ぎたあたりで収束していることが図から分かる。誤差もかなり小さいので、理想的に最適値を求められたと考えられる。



- 次元数: 20, サンプル数: 12000, ステップ数: 500
  50 ステップを過ぎたあたりで収束していることが図から分かる。20 次元とはいえ、誤差が小さいわけではないので、局所最小値に陥った可能性がある。

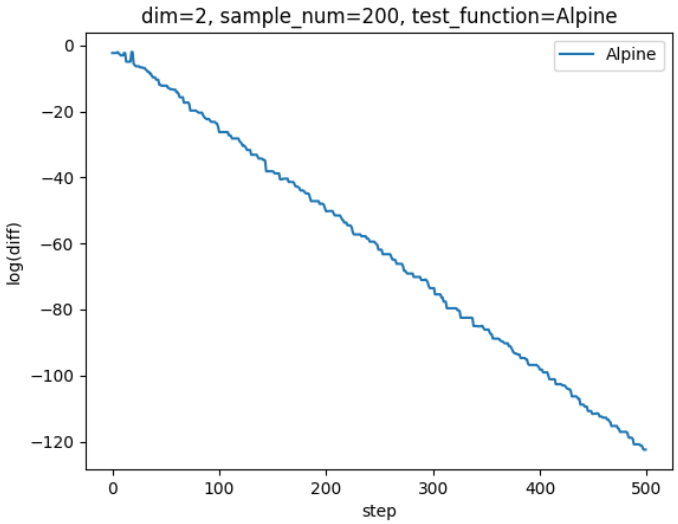


5. Alpine 関数
   関数の係数等は以下の設定で行った。

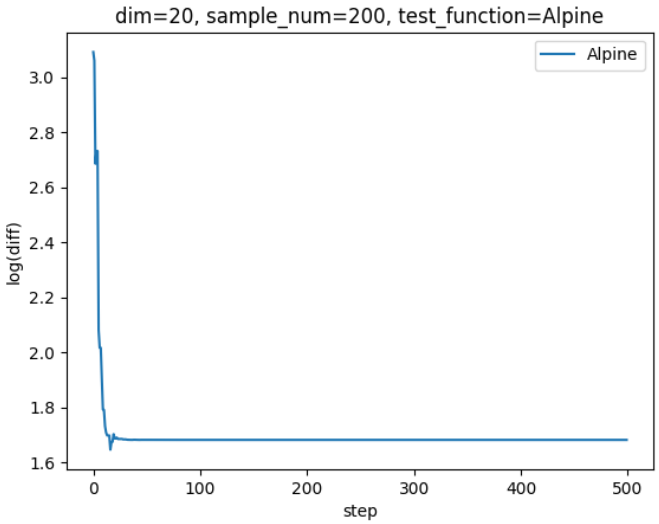
$$f(x) = \sum_{i=1}^{n} |x_i \sin(x_i) + 0.1x_i|, (-10.00 \leq x_i \leq 10.00) \tag{5}$$

この場合、最小値は全次元 0 の x を与えた際の 0 である。
以下に、次元数とサンプル数を変化させた際の結果を示す。グラフは、各ステップにおける最適解との差の対数をプロットしたものである。

- 次元数: 2, サンプル数: 200, ステップ数: 500
  図から分かるようにステップを経るにつれて最適解との差の対数が単調にほぼ一様に小さくなっている。収束した様子は見られず、ステップ数を増加させれば収束した可能性がある。また、非常に最適値と近い値をとっていることがわかる。



- 次元数: 20, サンプル数: 200, ステップ数: 500
  数十ステップで収束したと思われる。誤差の大きさからすると、20 次元とはいえ、誤差が小さいわけではないので、局所最小値に陥った可能性がある。



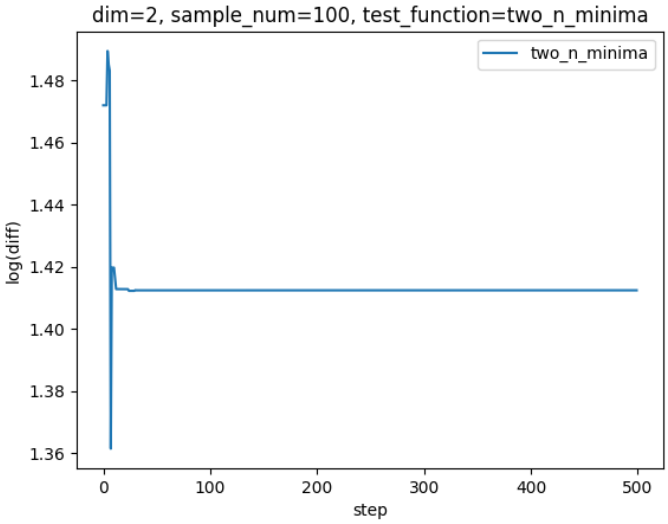
6. 2n-minima 関数
   関数の係数等は以下の設定で行った。

$$f(x) = \sum_{i=1}^{n}(x_i^4 - 16x_i^2 + 5x_i), (-5.00 \leq x_i \leq 5.00) \tag{6}$$

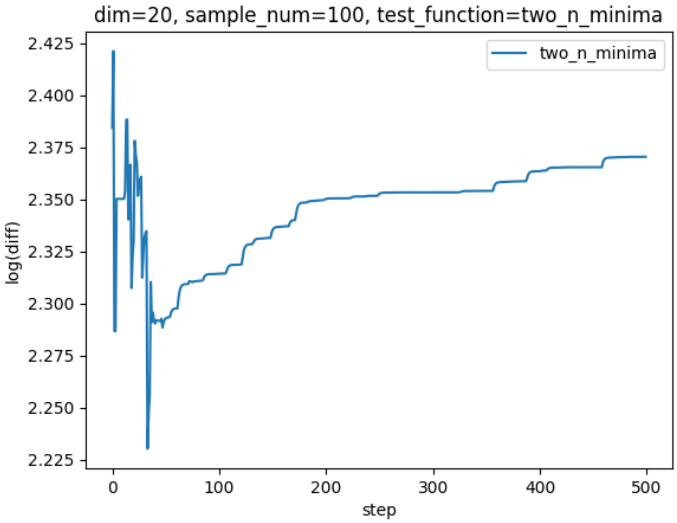
この場合、最小値は全次元 0 の x を与えた際の 0 である。
以下に、次元数とサンプル数を変化させた際の結果を示す。グラフは、各ステップにおける最適解との

差の対数をプロットしたものである。

- 次元数: 2, サンプル数: 100, ステップ数: 500
  一度急速に最適値へと近づいたものの、再び誤差が大きくなってから数十ステップで収束したと思われる。各ステップでの変動幅が大きかったせいで最適値を通り越してしまって局所最小値に陥ったと思われる。



- 次元数: 20, サンプル数: 100, ステップ数: 500
  二度急速に最適値へと近づいたものの、その後は誤差が大きくなり続けて収束しないまま 500 ステップが終了している。ステップ数を増やしても局所最小値で収束する可能性はあるものの、最適値で収束する可能性は低いと考えられる。



2 プログラムコード

プログラムは、pso.py, test_functions.py, main.py の 3 つのファイルから構成される。pso.py は、PSO のアルゴリズムを実装したものである。test_functions.py は、PSO の性能を評価するための関数を実装したものである。main.py は、PSO の性能を評価するためのプログラムである。main.py は、コマンドライン引数として、設定ファイルのパスを受け取る。設定ファイルは、yaml 形式で記述されている。以下には python のコードのみを示し、設定ファイルに関しては割愛する。
全てのコードは、github にて公開している。

Listing 1 pso.py

```python
import random
from typing import List

import numpy as np


def update_position(x: np.ndarray, v: np.ndarray) -> np.ndarray:
    updated_x = x + v
    return updated_x


def update_velosity(
    x: np.ndarray,
    v: np.ndarray,
    personal_best_position: np.ndarray,
    global_best_position: np.ndarray,
    w=0.5,
    c1=0.14,
    c2=0.14,
) -> np.ndarray:
    r1 = random.uniform(0, 1)
    r2 = random.uniform(0, 1)
    updated_v = (
        w * v
        + c1 * r1 * (personal_best_position - x)
        + c2 * r2 * (global_best_position - x)
    )
    return updated_v


def pso(
    step_num: int,
    initial: List[np.ndarray], # 各要素にその点の全次元位置情報が含まれる
```

```
        criterion,
    ) -> dict:
        # initialize
        num = len(initial)
        position_list = initial
        velosity_list = np.zeros((len(initial), len(initial[0])))
        personal_best_position_list = list(position_list)
        personal_best_score_list = [criterion(p) for p in position_list]
        best_particle_idx = np.argmin(personal_best_score_list)
        global_best_position = personal_best_position_list[best_particle_idx]

        result = {}

        for t in range(step_num):
            for i in range(num):
                x = position_list[i]
                v = velosity_list[i]
                personal_best_position = personal_best_position_list[i]

                updated_x = update_position(x, v)
                position_list[i] = updated_x

                updated_v = update_velosity(
                    x, v, personal_best_position, global_best_position
                )
                velosity_list[i] = updated_v

                score = criterion(updated_x)
                if score < personal_best_score_list[i]:
                    personal_best_score_list[i] = score
                    personal_best_position_list[i] = updated_x

            best_particle_idx = np.argmin(personal_best_score_list)
            global_best_position = personal_best_position_list[best_particle_idx]

            result[t] = {
                "global_best_position": global_best_position,
                "global_best_score": min(personal_best_score_list),
            }

        return result
```

Listing 2 test functions.py

```
import numpy as np


def Sphere(x: np.ndarray) -> np.ndarray:
    return np.sum(x**2)


def Rastrigin(x: np.ndarray) -> np.ndarray:
    return np.sum(x**2 - 10 * np.cos(2 * np.pi * x) + 10)


def Rosenbrock(x: np.ndarray) -> np.ndarray:
    return np.sum(100 * (x[1:] - x[:-1] ** 2) ** 2 + (1 - x[:-1]) ** 2)


def Griewank(x: np.ndarray) -> np.ndarray:
    return (
        np.sum(x**2 / 4000)
        - np.prod(np.cos(x / np.sqrt(np.arange(1, len(x) + 1))))
        + 1
    )


def Alpine(x: np.ndarray) -> np.ndarray:
    return np.sum(np.abs(x * np.sin(x) + 0.1 * x))


def two_n_minima(x: np.ndarray) -> np.ndarray:
    return np.sum(x**4 - 16 * x**2 + 5 * x)
```

Listing 3 main.py

```
import sys

import matplotlib.pyplot as plt
import numpy as np
import yaml
from pso import pso
from test_functions import Alpine, Griewank, Rastrigin, Rosenbrock, Sphere,
    two_n_minima


def main(config_file_path: str):
    with open(config_file_path) as file:
        config = yaml.safe_load(file)

    dim = config["dim"]
    sample_num = config["sample_num"]
    low = config["low"]
    high = config["high"]
    step_num = config["step_num"]

    initial = []
    for i in range(sample_num):
        initial.append(np.random.uniform(low, high, dim))

    test_function = config["test_function"]
    if test_function == "Sphere":
        test_function = Sphere
    elif test_function == "Rastrigin":
        test_function = Rastrigin
    elif test_function == "Rosenbrock":
        test_function = Rosenbrock
    elif test_function == "Griewank":
        test_function = Griewank
    elif test_function == "Alpine":
        test_function = Alpine
    elif test_function == "two_n_minima":
        test_function = two_n_minima
    else:
        raise ValueError("test_function is invalid")

    ideal_position = np.zeros(dim)

    print(test_function.__name__)
    result = pso(step_num, initial, criterion=test_function)
    global_best_position_list = []
```

```
        diff_log_list = []
        for i in range(step_num):
            global_best_position_list.append(result[i]["global_best_position"])
            diff_log = np.log(np.linalg.norm(ideal_position - global_best_position_list[
                i]))
            diff_log_list.append(diff_log)
            print(f"step {i}: {diff_log}")
        plt.plot(diff_log_list, label=test_function.__name__)
        plt.xlabel("step")
        plt.ylabel("log(diff)")
        plt.title(
            f"dim={dim}, sample_num={sample_num}, test_function={test_function.__name__
                }"
        )
        plt.legend()
        plt.savefig(
            f"./result/{test_function.__name__}_N_{dim}.png"
        )
        print("----------------------")


    if __name__ == "__main__":
        main(sys.argv[1])
```