令和 7 年度

実 験 レ ポ ー ト

題目	進化計算アルゴリズム

学　　科	電気電子システム工学コース
学籍番号	a0527
氏　　名	野口 史遠
提 出 日	令和 7 年 6 月 30 日

National Institute of Technology, Maizuru College
NIT, Maizuru KYOTO

舞鶴工業高等専門学校

目 次

実験目的

工学領域における多くの問題は最適化問題として定式化することができ，これに対する解法としてさまざまな手法が研究・開発されている．従来，最適化問題を解くための理論や技術は，数学的な理論に基づく最適化理論や数理計画法といった研究分野で発展してきた．しかし，現実世界の問題には，数学的に解析して解くことが非常に難しい問題や，限られた時間内で最適解を求めることが困難な問題も数多く存在している．このようなことから，こうした現実的な制約のある問題に対しては，進化計算と呼ばれる手法が効果的な解法の一つとして広く利用されるようになった．進化計算の代表的な手法としては，Holland によって提案された遺伝的アルゴリズム（Genetic Algorithm, GA）や，Kennedy と Eberhart によって提案された粒子群最適化（Particle Swarm Optimization, PSO）などがある．これらの手法は，自然界の進化の仕組みや生物の振る舞いをモデルとした手法である [1]．本実験では，これらの進化計算アルゴリズムの理解を深めるため，基本的なベンチマーク問題を用いた数値実験を Python を用いて実装したプログラムを用いて行う．また，進化計算の工学設計問題の応用として圧力容器設計への応用についても数値実験を行う．

第 1 章　数値実験

1.1　課題 1

1.1.1　課題内容

最適化アルゴリズムの評価には，さまざまなテスト関数が利用されている．**Rastrigin 関数以外**の単一目的関数を調査し，それらの特徴を整理して Python で実装しなさい．

文献 [2] などを参考にし，最適化アルゴリズムのベンチマーク関数としてよく使用されるテスト関数について調査する．

なお，テスト関数の性質としては以下のような要素がある．

- **単峰性関数（Unimodal functions）**：最適解がただ一つの谷に存在する関数
- **多峰性関数（Multimodal functions）**：最適解が複数の谷に存在する関数
- **変数間の依存関係の有無（Variable dependencies）**：変数が独立しているか，または相互に依存しているか

具体的なテスト関数として，次に示す 3 つの関数を紹介する．これらの関数は最適化問題において広く利用され，それぞれ異なる性質を持っている．

- **Sphere 関数**：この関数は，すべての変数が独立しており，最適解は原点（全ての変数が 0）にある．

$$f_{\text{Sphere}}(x_i) = \sum_{i=1}^{D} x_i^2, \quad (-10 \leq x_i \leq 10) \tag{6}$$

- **Rosenbrock 関数（Rosenbrock-Star）**：この関数は，変数間に依存関係がある．

$$f_{\text{Rosenbrock-Star}}(x_i) = \sum_{i=1}^{D-1} \left\{ 100(x_{i+1} - x_i^2)^2 + (x_i - 1.0)^2 \right\}, \quad (-10 \leq x_i \leq 10) \tag{7}$$

- **Griewank 関数**：この関数は，複数の変数が相互に影響し合う形式で，非常に多くの局所最小値を持っている．

$$f_{\text{Griewank}}(x_i) = \frac{1}{4000} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1, \quad (-500 \leq x_i \leq 500) \tag{8}$$

各関数の特徴を理解し，それぞれ Python で実装する．そして，実装したテスト関数を利用し，PSO で最適化を行いなさい．

1.1.2　課題結果及び考察

本課題では，最適化アルゴリズムである粒子群最適化（PSO）を用いて，4 種類のベンチマーク関数（Sphere 関数，Rosenbrock 関数，Griewank 関数，Rastrigin 関数）に対して最小値探索を行い，それぞれの収束過程を比較・考察した．

図 1.1 に各関数における最良評価値の世代推移を示す．Sphere 関数は最も単純な単峰性かつ独立変数型の関数であり，PSO は初期世代から急速に最適解へと収束した．Rastrigin 関数および Griewank 関数は多峰性関数であり，多数の局所解を持つにもかかわらず，PSO はそれらを回避して比較的スムーズに収束した．特に Griewank 関数では，非常に滑らかな収束曲線が得られており，PSO の探索性能の高さが伺える．

一方，Rosenbrock 関数は変数間に強い依存関係を持ち，評価関数の谷が細く曲がっていることから，PSO のような群知能アルゴリズムにとっては収束が困難であることが知られている．本実験においても，初期世代で高い評価値を示し，収束に時間を要していることが確認できた．これは PSO が探索空間の谷に沿って進むのに苦戦している様子を反映していると考えられる．

以上より，PSO は多峰性関数に対しても一定の有効性を示すが，変数間に依存関係がある関数に対しては収束性に課題が残ることが示された．今後は局所探索を補強する手法（例：ハイブリッドアルゴリズム）やパラメータチューニングにより，収束性能の向上が期待される．

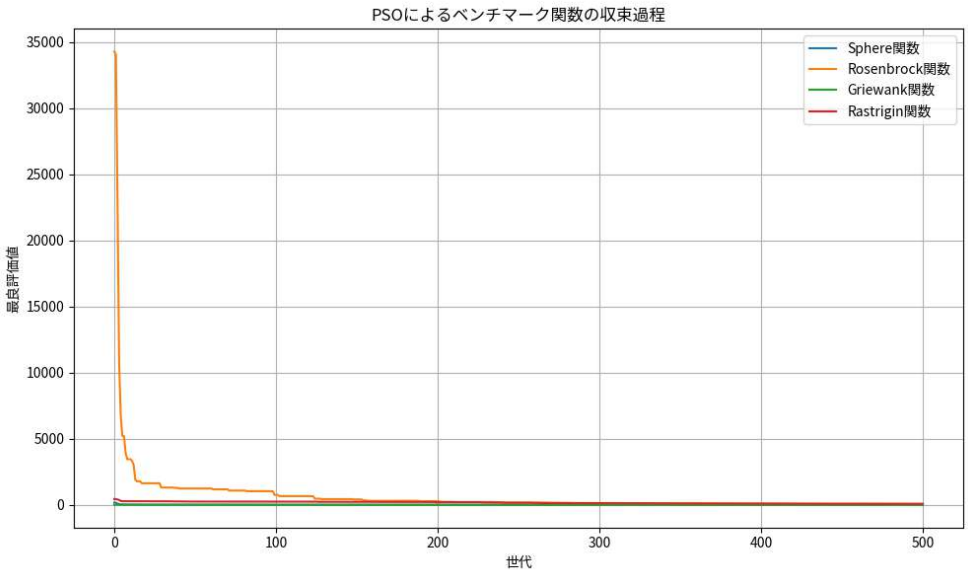

図 1.1 ：PSO によるベンチマーク関数の収束過程

1.1.3　使用したプログラムコード

```
import os
import math
import random
import matplotlib.pyplot as plt
import pandas as pd

# 日本語フォント設定
plt.rcParams['font.family'] = 'Noto Sans CJK JP'

def fitFuncSphere(xVals):
    return sum(x**2 for x in xVals)

def fitFuncRosenbrock(xVals):
    return sum(100 * (xVals[i] - xVals[i-1])**2 + (xVals[i-1] - 1)**2 for i in range(1, len(xVals)))

def fitFuncGriewank(xVals):
    sum_term = sum(x**2 for x in xVals) / 4000
    prod_term = 1
    for i in range(len(xVals)):
        prod_term *= math.cos(xVals[i] / math.sqrt(i + 1))
    return sum_term - prod_term + 1

def fitFuncRastrigin(xVals):
    return 10 * len(xVals) + sum(x**2 - 10 * math.cos(2 * math.pi * x) for x in xVals)

def initPosition(Np, Nd, xMin, xMax):
    return [[xMin + random.random() * (xMax - xMin) for _ in range(Nd)] for _ in range(Np)]

def initVelocity(Np, Nd, vMin, vMax):
    return [[vMin + random.random() * (vMax - vMin) for _ in range(Nd)] for _ in range(Np)]

def updateVelocity(R, V, Np, Nd, j, w, vMin, vMax, pBestPos, gBestPos, c1, c2):
    for p in range(Np):
        for i in range(Nd):
            r1 = random.random()
            r2 = random.random()
            V[p][i] = (w * V[p][i] +
                       c1 * r1 * (pBestPos[p][i] - R[p][i]) +
                       c2 * r2 * (gBestPos[i] - R[p][i]))
            V[p][i] = max(min(V[p][i], vMax), vMin)

def updatePosition(R, V, Np, Nd, xMin, xMax):
    for p in range(Np):
        for i in range(Nd):
            R[p][i] += V[p][i]
            R[p][i] = max(min(R[p][i], xMax), xMin)

def updateFitness(R, M, Np, pBestPos, pBestVal, gBestPos, gBestValue, fitFunc):
    for p in range(Np):
        M[p] = fitFunc(R[p])
        if M[p] < gBestValue:
            gBestValue = M[p]
            gBestPos[:] = R[p][:]
        if M[p] < pBestVal[p]:
            pBestVal[p] = M[p]
            pBestPos[p][:] = R[p][:]
    return gBestValue
```

```
58
59  def runPSO(fitFunc, name, Np=30, Nd=30, Nt=500,
60            xMin=-5.12, xMax=5.12,
61            vMin=None, vMax=None,
62            wMin=0.4, wMax=0.9, c1=2.05, c2=2.05):
63
64      vMin = vMin if vMin is not None else 0.25 * xMin
65      vMax = vMax if vMax is not None else 0.25 * xMax
66
67      R = initPosition(Np, Nd, xMin, xMax)
68      V = initVelocity(Np, Nd, vMin, vMax)
69      M = [fitFunc(R[p]) for p in range(Np)]
70      pBestVal = M[:]
71      pBestPos = [r[:] for r in R]
72      gBestValue = min(M)
73      gBestPos = R[M.index(gBestValue)][:]
74      history = [gBestValue]
75
76      for j in range(Nt):
77          w = wMax - (wMax - wMin) * j / Nt
78          updatePosition(R, V, Np, Nd, xMin, xMax)
79          gBestValue = updateFitness(R, M, Np, pBestPos, pBestVal, gBestPos, gBestValue, fitFunc)
80          updateVelocity(R, V, Np, Nd, j, w, vMin, vMax, pBestPos, gBestPos, c1, c2)
81          history.append(gBestValue)
82
83      return name, history
84
85  def main():
86      output_dir = "実験結果./1"
87      os.makedirs(output_dir, exist_ok=True)
88
89      benchmark_funcs = [
90          (fitFuncSphere, "関数 Sphere"),
91          (fitFuncRosenbrock, "関数 Rosenbrock"),
92          (fitFuncGriewank, "関数 Griewank"),
93          (fitFuncRastrigin, "関数 Rastrigin")
94      ]
95
96      all_histories = []
97      df_data = {}
98
99      for func, name in benchmark_funcs:
100         print(f"{name} の最適化を実行中 PSO...")
101         name, history = runPSO(func, name)
102         all_histories.append((name, history))
103         df_data[name] = history
104
105     df = pd.DataFrame(df_data)
106     df.index.name = "世代"
107     df.to_csv(os.path.join(output_dir, "収束履歴表.csv"))
108
109     plt.figure(figsize=(10, 6))
110     for name, history in all_histories:
111         plt.plot(history, label=name)
112     plt.xlabel("世代")
113     plt.ylabel("最良評価値")
114     plt.title("によるベンチマーク関数の収束過程 PSO")
115     plt.legend()
116     plt.grid(True)
```

```
    plt.tight_layout()
    plt.savefig("figure収束グラフ/.pdf")
    plt.show()

if __name__ == "__main__":
    main()
```

Code 1: PSO によるベンチマーク関数の最適化

1.2 課題 2

1.2.1 課題内容

1.2.2 課題結果及び考察

1.3 課題 3

1.3.1 課題内容

1.3.2 課題結果及び考察

1.4 課題 4

1.4.1 課題内容

1.4.2 課題結果及び考察

1.5 課題 5

1.5.1 課題内容

1.5.2 課題結果及び考察

1.6 課題 6

1.6.1 課題内容

1.6.2 課題結果及び考察

1.7 課題 7

1.7.1 課題内容

1.7.2 課題結果及び考察

参 考 文 献

1) 相吉英太郎, 安田恵一郎編著, 『メタヒューリスティクスと応用』, 電気学会, 2007.

2) Axel Thevenot, “Optimization & Eye Pleasure: 78 Benchmark Test Functions for Single Objective Optimization”, https://github.com/AxelThevenot/Python_Benchmark_Test_Optimization_Function_Single_Objective

3) Google Cloud: Google Maps Platform Documentation, https://developers.google.com/maps/documentation（参照日：2025 年 6 月 8 日）.