# GA & PSO 演習報告

**ID: 20202251201　Name: 陳実**

## 1. 序論

遺伝的アルゴリズム（GA）は、自然進化過程をシミュレーションすることによって最適解を探索するアルゴリズムである。複雑な組合せ最適化問題を解く場合、伝統的な最適化アルゴリズムと比べて、より良い最適化結果を迅速に得ることができる。しかし、遺伝的アルゴリズムは通常、伝統的な最適化アルゴリズムより効率が低い。

粒子群最適化法（PSO）は、集団協調に基づくランダム探索アルゴリズムである。かなり速い近似最適解の速度を持ち、効率的にシステムのパラメータを最適化することができる。しかし、いくつかの問題では、性能が良くない。

今回の演習では、遺伝的アルゴリズムと粒子群最適化法を実現し、目標関数：

$$y = 2x_1^2 - 3x_2^2 - 4x_1 + 5x_2 + x_3$$

の最大化及び最小化を求める。**最大化の結果は 407.08333、最小化の結果は-602.00000。**私は Python と VSCode で遺伝的アルゴリズムと粒子群最適化法を実現し、matplotlib を利用して結果の画像を表現した。**完全なソースコードは py ファイルにある。**第 2 部分では、遺伝的アルゴリズムの部分ソースコードと結果の画像を示す。第 3 部分では、粒子群最適化法のを示す。最後に第 4 部分では、簡単な結論である。

# 2. 遺伝的アルゴリズム

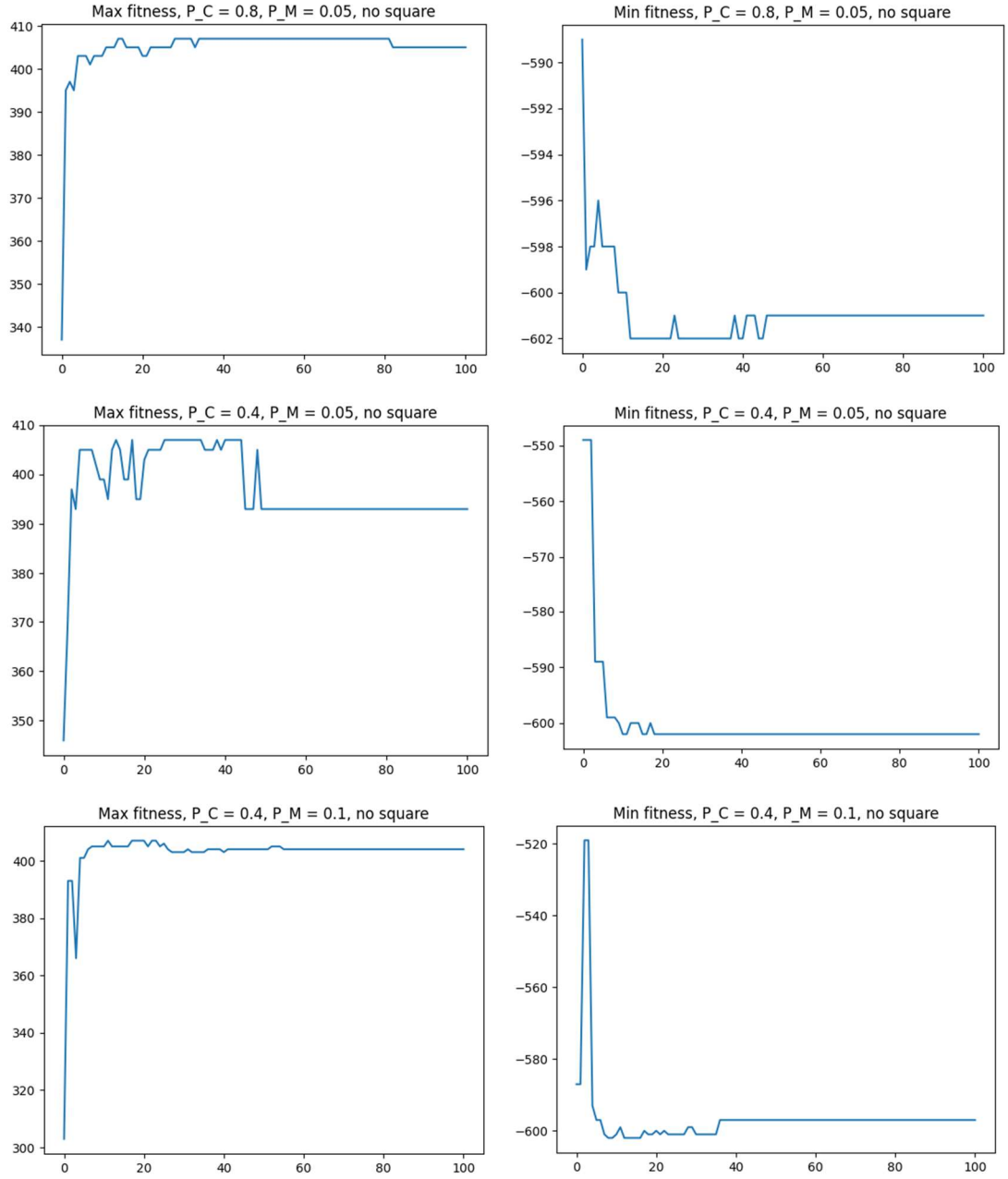
## 2.1. 部分ソースコード

詳細は Genetic_Algorithm.py を参照。

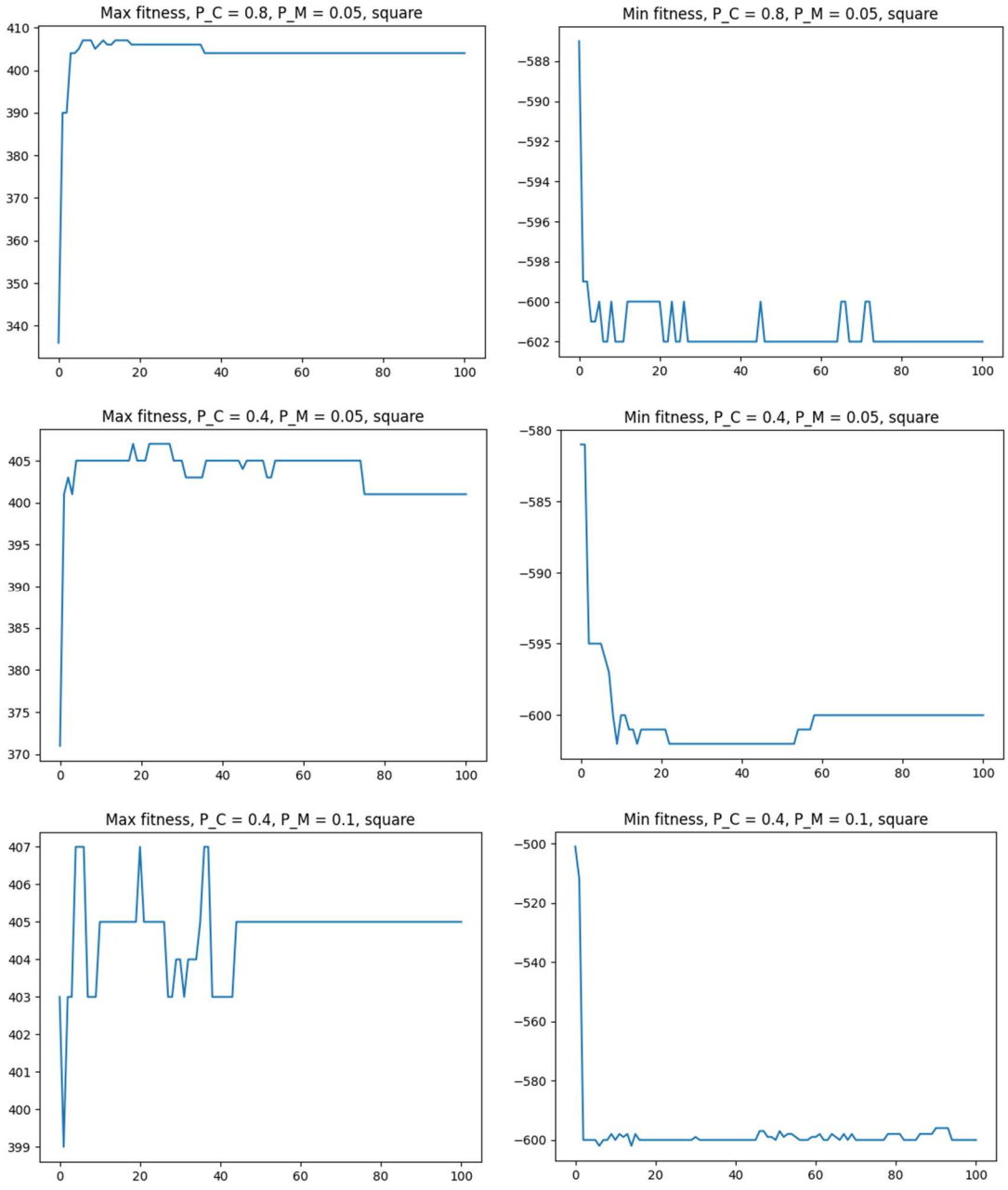
```
# GA_function
def GA_function(max_or_min, is_square):
# generation 0
individuals = np.random.randint(2, size=(NUM, GENE_LENGTH * 3))
# calculate fitness
fitness = calculate_fitness(individuals, max_or_min)
# 1~T generations
for _ in range(T):
# roulette
individuals = roulette_selection(individuals, fitness, is_square)
# crossover & mutation
individuals = np.array(mutation(crossover(individuals)))
# calculate fitness
fitness = calculate_fitness(individuals, max_or_min)
# print max or min fitness
if max_or_min == 1:
print("max fitness: %.5f" % (np.max(MAX_FITNESS)))
elif max_or_min == -1:
print("min fitness: %.5f" % (np.min(MIN_FITNESS)))

if __name__ == "__main__":
# get max fitness
GA_function(1, IS_SQUARE)
# get min fitness
GA_function(-1, IS_SQUARE)
# show the result by graph
plt.plot(MAX_FITNESS)
# set titles
plt.title("Max fitness, P_C = " + str(P_C) + ", P_M = " + str(P_M) +
(", square" if IS_SQUARE == 1 else ", no square"))
plt.show()
plt.plot(MIN_FITNESS)
plt.title("Min fitness, P_C = " + str(P_C) + ", P_M = " + str(P_M) +
(", square" if IS_SQUARE == 1 else ", no square"))
plt.show()
```

## 2.2. 結果

まずは結果の画像：




上の 6 つの図は、適応度に比例させる場合に、パラメータを調整する結果の画像である。*P_C* は交叉確率であり、*P_M* は突然変異確率である。交叉確率は低いと、個体群の更新が遅くなるが、高いと、既存の最適解を破壊しやすく、最適解のランダム性が増大する。突然変異確率は低いと、収束不安定を招きやすいが、高いと、最適解のランダム性が増大する。




上の 6 つの図は、適応度の 2 乗に比例させる場合に、パラメータを調整する結果の画像である。適応度の 2 乗に比例させる場合、適応度に比例させる場合より、最適解が選択される確率が高くなり、得られやすくなる。

# 3. 粒子群最適化法

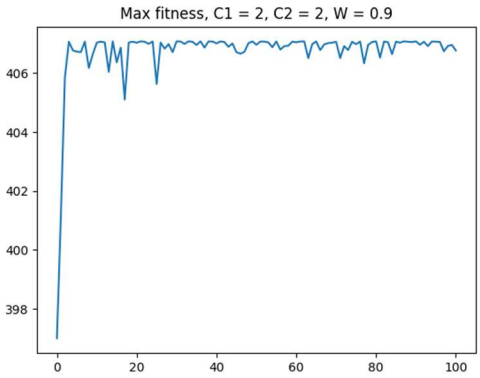
## 3.1. 部分ソースコード

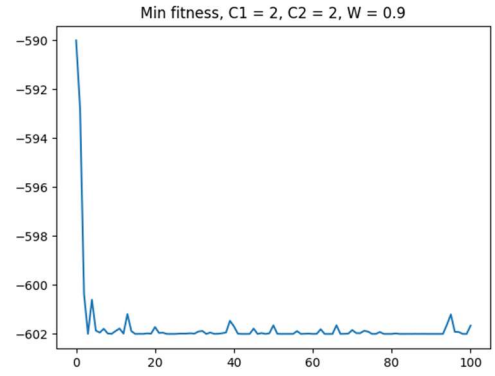
詳細は Particle_Swarm_Optimization.py を参照。

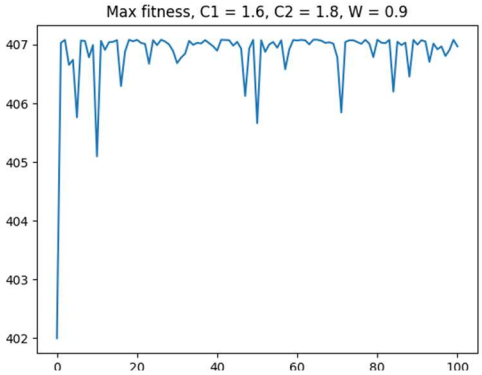
```
# PSO function
def PSO_function(self, max_or_min):
for _ in range(self.max_steps):
# according to the formula
# r1 and r2 are random numbers, increasing the search randomness
r1 = np.random.rand(self.individual_size, self.dim)
r2 = np.random.rand(self.individual_size, self.dim)
# update velocity and C1, C2
self.v = W*self.v+C1*r1*(self.p-self.x)+C2*r2*(self.pg-self.x)
minn = [[self.x_bound[0]]*self.dim]*self.individual_size
maxn = [[self.x_bound[1]]*self.dim]*self.individual_size
# limit the bound
self.x = clamp(self.v + self.x, minn, maxn)
# calculate fitness
fitness = self.calculate_fitness(self.x, max_or_min)
# update one individual
update_id = np.greater(self.individual_best_fitness, fitness)
self.p[update_id] = self.x[update_id]
self.individual_best_fitness[update_id] = fitness[update_id]
# update global best fitness
if np. max(fitness) > self.global_best_fitness:
self.pg = self.x[np.argmax(fitness)]
self.global_best_fitness = np.max(fitness)
# append into the sets
if max_or_min == 1:
MAX_FITNESS.append(np.max(fitness))
elif max_or_min == -1:
MIN_FITNESS.append(-np.max(fitness))
```

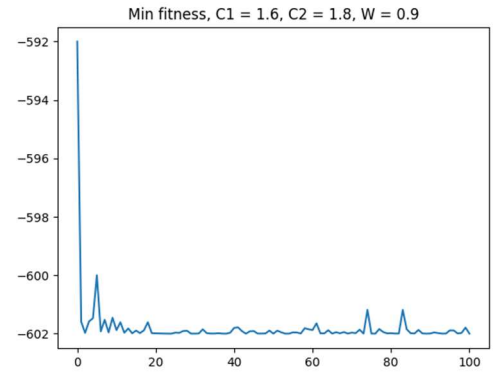
## 3.2. 結果

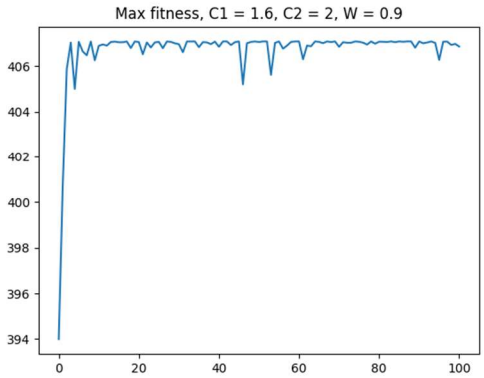
まずは結果の画像：

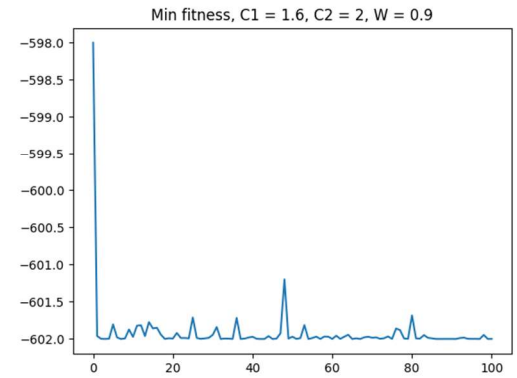


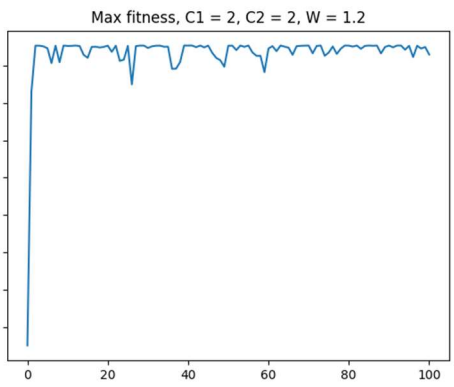



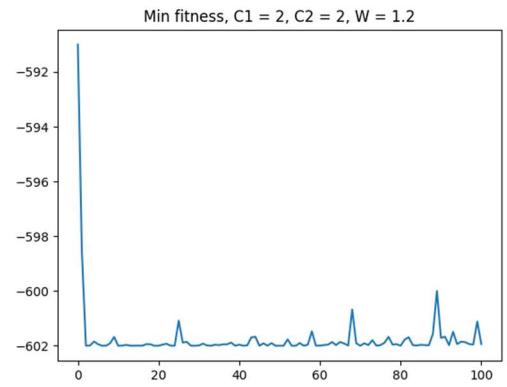



上の 8 つの図は、諸パラメータを調整する結果の画像である。*C1*、*C2* は学習係数、*W* は慣性係数である。*W* は大きいと、グローバル検索に有利であり、局所最適に陥りにくいが、小さいと、局所探索に有利であり、最適解に迅速に収束させる。*C1* は小さいと、集団の多様性を失い、局所最適に陥りやすく飛び出せない。*C2* は小さいと、アルゴリズムの収束速度は遅くなる。

## 4. 結論

2 つのアルゴリズムを比較すると、GA では記憶がなく、以前の知識は個体群の変化に伴って破壊される。PSO では記憶があり、優解の知識は全ての粒子が保存されている。両者の情報共有メカニズムも異なる：GA では、染色体は互いに情報を共有し、個体群全体の移動は最適領域へ均一に移動するが、PSO では、*gBest* だけが他の粒子に情報を与え、これは一方向の情報フローである。GA と比較して、ほとんどの場合、PSO の粒子はより早く最適解に収束する可能性がある。