計算知能レポート

学籍番号：201894086

氏名：黄彬倩

一．遺伝的アルゴリズム

1.グローバル変数を宣言する

```
N = 15 #遺伝子長
Pc = 0.30 #交叉確率
Pm = 0.05 #突然変異確率
N_GENERTATIONS = 100 #世代数
POP_SIZE = 200 #群の行列
X1_BOUND = [0, 15] #x1取値範囲
X2_BOUND = [0, 15] #x2取値範囲
X3_BOUND = [0, 15] #x3取値範囲
```

2.関数を実装する

•求めたい y

```
def F(x1, x2, x3):
    return 2 * x1 * x1 - 3 * x2 * x2 - 4 * x1 + 5 * x2 + x3;
```

•復号プロセス

```
def translateDNA(pop): #popは群行列を表し、行はバイナリ符号化で表されるDNA
                       #を表し、行列の行数は群の数である。
    x1_pop = pop[:, :N]
    x2_pop = pop[:, N:-N]
    x3_pop = pop[:, -N:]
    x1 = x1_pop.dot(2 ** np.arange(N)[::-1]) / float(2 ** N - 1) * \
        (X1_BOUND[1] - X1_BOUND[0]) + X1_BOUND[0]
    x2 = x2_pop.dot(2 ** np.arange(N)[::-1]) / float(2 ** N - 1) * \
        (X2_BOUND[1] - X2_BOUND[0]) + X2_BOUND[0]
    x3 = x3_pop.dot(2 ** np.arange(N)[::-1]) / float(2 ** N - 1) * \
        (X3_BOUND[1] - X3_BOUND[0]) + X3_BOUND[0]
    return x1, x2, x3
```

•最大値適応度計算関数

環境に対する個人の適応度を評価し、最大値を求める問題では、解（個体）に対応する関数の大きさで直接評価でき、対応する関数の値が大きいほど保持される可能性があります。pred は、関数 F に展開可能な予測値です。後者の選択過程は、個々の適応度に基づいて個々の個体が保持される確率を決定する必要があります。確率は負の値ではないので、予測中の最小値を減算して、適応度値の最小区間を 0 から開始します。

```
def get_fitness(pop):
    x1, x2, x3 = translateDNA(pop)
    pred = F(x1, x2, x3)
    return pred - np.min(pred)#最小の適応度を減算するのは、適応度に負が生じるのを防ぐ
                              #ためfitnessの範囲は[0,np.max(pred)-np.min(pred)
```

•最小値適応度計算関数

```
def get_fitness(pop):
    x, y = translateDNA(pop)
    pred = F(x, y)
    return -(pred - np.max(pred))
```

•一点交叉関数

それぞれの個体は父と母の 2 つの個体が繁殖して生まれ、子世代の DNA は半分の父の DNA を獲得しました。半分の母の DNA ですが、ここの半分は本当の半分ではありません。この位置は交配点といいます。ランダムに発生したもので、DNA の任意の位置です。

```
def crossover(pop, rate=Pc):
    new_pop = []
    for father in pop:# 群れの中のすべての個体を巡り、その個体を父親とする
        child = father# 子供は父の遺伝子をもらう
        if np.random.rand() < rate:# 一定の確率で交差が発生する
            mother = pop[np.random.randint(POP_SIZE)]# もう一つの个体を母
                                                    # としてえらぶ
            cross_points = np.random.randint(low=0, high=N * 3)# ランダムに交差点
                                                               # を生成する
            child[cross_points:] = mother[cross_points:]# 子供は交差点の後ろにある
                                                        # 母の遺伝子をもらう
        mutation(child)# 子供ごとに一定の確率で変異が起こる。
        new_pop.append(child)
    return new_pop
```

•突然変異関数

子供自身が変異を起こす可能性があります。DNA は父でも母でもないので、ある位置でランダムに変化します。通常は DNA を変えるバイナリビットです。

```
def mutation(child, rate=Pm):
    if np.random.rand() < rate:# rateの確率で変異する
        mutate_point = np.random.randint(0, N * 3)# ランダムに実数を生成し、遺伝子を
                                                  # 変異させる位置を表す
        child[mutate_point] = child[mutate_point] ^ 1# 変異点のバイナリを反転しする
```

•群を生成する関数

選択は新しい個体の適応度によって行われるが、同時に完全に適応度の高低をガイドにするという意味ではなく、単純に適応度の高い個体を選ぶとアルゴリズムが大域最適解

ではなく局所最適解に急速に収束する可能性があるからである。遺伝的アルゴリズムは原則によって適応度が高いほど、選択される機会が高く、適応度が低いほど、選択される機会が低い。

```
def select(pop, fitness):# 群を生成する
    idx = np.random.choice(np.arange(POP_SIZE), size=POP_SIZE,
                           replace=True, p=(fitness) / (fitness.sum()))
    return pop[idx]
```

•最大値計算関数

```
def print_pre(pop):
    fitness = get_fitness(pop)
    max_fitness_index = np.argmax(fitness)
    fit=round(fitness[max_fitness_index],2)
    print("max_fitness", fit)
    x1, x2, x3 = translateDNA(pop)
    y=F(x1[max_fitness_index], x2[max_fitness_index], x3[max_fitness_index])
    y1=round(y,2)
    print("2*x1*x1-3*x2*x2-4*x1+5*x2+x3=",y1)
```

•最小値計算関数

```
def print_pre(pop):
    fitness = get_fitness(pop)
    min_fitness_index = np.argmin(fitness)
    fit=round(fitness[min_fitness_index], 2)
    print("min_fitness", fit)
    x1, x2, x3 = translateDNA(pop)
    y=F(x1[min_fitness_index], x2[min_fitness_index], x3[min_fitness_index])
    y1=round(y,2)
    print("2*x1*x1-3*x2*x2-4*x1+5*x2+x3=",y1)
```

•main 関数

```
if __name__ == '__main__':
    pop = np.random.randint(2, size=(POP_SIZE, N * 3))# 群を生成する
    print(pop)
    for i in range(N_GENERTATIONS):# 群は反復して進化するN_GENERATIONS代
        x1, x2, x3 = translateDNA(pop)
        pop = np.array(crossover(pop))# 交差変異を通して後代を生む
        fitness = get_fitness(pop)# 群の中の各個体を評価する
        pop = select(pop, fitness)# 新しいグループを作成する場合にする
        print("{:=^50}\n{}番目".format("", i + 1))
        print_pre(pop)
```

3. 出力

- 最大値

```
1番目
max_fitness 954.15
2*x1*x1-3*x2*x2-4*x1+5*x2+x3= 386.79
==================================================
2番目
max_fitness 664.3
2*x1*x1-3*x2*x2-4*x1+5*x2+x3= 386.79
==================================================
3番目
max_fitness 629.54
2*x1*x1-3*x2*x2-4*x1+5*x2+x3= 386.89
==================================================
4番目
max_fitness 427.67
2*x1*x1-3*x2*x2-4*x1+5*x2+x3= 386.58
==================================================
5番目
max_fitness 427.72
2*x1*x1-3*x2*x2-4*x1+5*x2+x3= 386.69
==================================================
6番目
max_fitness 390.01
2*x1*x1-3*x2*x2-4*x1+5*x2+x3= 386.78
==================================================
7番目
max_fitness 332.43
2*x1*x1-3*x2*x2-4*x1+5*x2+x3= 386.78
==================================================
```

ここは省略します。

```
94番目
max_fitness 14.84
2*x1*x1-3*x2*x2-4*x1+5*x2+x3= 406.55
==================================================
95番目
max_fitness 14.84
2*x1*x1-3*x2*x2-4*x1+5*x2+x3= 406.55
==================================================
96番目
max_fitness 7.57
2*x1*x1-3*x2*x2-4*x1+5*x2+x3= 406.55
==================================================
97番目
max_fitness 7.35
2*x1*x1-3*x2*x2-4*x1+5*x2+x3= 406.55
==================================================
98番目
max_fitness 7.74
2*x1*x1-3*x2*x2-4*x1+5*x2+x3= 406.34
==================================================
99番目
max_fitness 5.94
2*x1*x1-3*x2*x2-4*x1+5*x2+x3= 406.35
==================================================
100番目
max_fitness 9.5
2*x1*x1-3*x2*x2-4*x1+5*x2+x3= 406.36

最大値：407

Process finished with exit code 0
```

•最小値

```
1番目
2*x1*x1-3*x2*x2-4*x1+5*x2+x3= 275.83
==================================================
2番目
2*x1*x1-3*x2*x2-4*x1+5*x2+x3= 152.26
==================================================
3番目
2*x1*x1-3*x2*x2-4*x1+5*x2+x3= 98.79
==================================================
```

```
4番目
2*x1*x1-3*x2*x2-4*x1+5*x2+x3= 32.48
==================================================
5番目
2*x1*x1-3*x2*x2-4*x1+5*x2+x3= -158.35
==================================================
6番目
2*x1*x1-3*x2*x2-4*x1+5*x2+x3= -185.67
==================================================
7番目
2*x1*x1-3*x2*x2-4*x1+5*x2+x3= -222.44
```

ここは省略します。

```
96番目
2*x1*x1-3*x2*x2-4*x1+5*x2+x3= -592.13
==================================================
97番目
2*x1*x1-3*x2*x2-4*x1+5*x2+x3= -592.13
==================================================
98番目
2*x1*x1-3*x2*x2-4*x1+5*x2+x3= -594.68
==================================================
99番目
2*x1*x1-3*x2*x2-4*x1+5*x2+x3= -592.0
==================================================
100番目
2*x1*x1-3*x2*x2-4*x1+5*x2+x3= -592.0
最小値：-599

Process finished with exit code 0
```

4. 評価

交叉と変異は必然的に起こるのではなく、一定の確率で起こるのです。まず交差を考慮して、最悪の場合、交差によって発生した子世代の DNA は父世代よりも悪くなります。（このように計算方は最適化の逆方向に向かって進行する可能性がありまして、収束しません。）、もし交差が一定の確率で発生しないなら、子世代の一部の遺伝子と現在の世代の基因レベルが同じであることを保証できます。変異は本質的にアルゴリズムを局所的な最適解から飛び出させることであり、変異が時々発生したり、発生確率が大きすぎると、アルゴリズムが最適解に到達した時には不安定になります。交差確率は、範囲は一般的に 0.6～1 で、変異確率は通常 0.1 以下です。

二．粒子群最適化法

1. 变数を宣言する

```
def __init__(self, c1=2, c2=2, w=2):
    self.w = w  # 慣性定数
    self.v0 = 0  # 探索個体の初速度
    self.v1 = 0
    self.v2 = 0
    self.x0 = random.randint(0, 15)  # 探索個体の位置
    self.x1 = random.randint(0, 15)
    self.x2 = random.randint(0, 15)
    self.y = calculate(self.x0, self.x1, self.x2)
    self.c1 = c1  # 学習係数
    self.c2 = c2
    self.p = [self.x0, self.x1, self.x2, self.y]  # 探索個体の最良位置
```

2. 探索関数 `def go(self, g):  # 探索関数`

•速度をもとめる

```
self.v0 = self.w * self.v0 + self.c1 * random.random() * (g[0] - self.x0) + \
          self.c2 * random.random() * (self.p[0] - self.x0)
self.v1 = self.w * self.v1 + self.c1 * random.random() * (g[1] - self.x1) + \
          self.c2 * random.random() * (self.p[1] - self.x1)
self.v2 = self.w * self.v2 + self.c1 * random.random() * (g[2] - self.x2) + \
          self.c2 * random.random() * (self.p[2] - self.x2)
```

•場所をもとめる

```
self.x0 = self.x0 + self.v0  # 限られた値の範囲
if self.x0 > 15:
    self.x0 = 15
elif self.x0 < 0:
    self.x0 = 0
self.x1 = self.x1 + self.v1
```

```
if self.x1 > 15:
    self.x1 = 15
elif self.x1 < 0:
    self.x1 = 0
self.x2 = self.x2 + self.v2
if self.x2 > 15:
    self.x2 = 15
elif self.x2 < 0:
    self.x2 = 0
self.y = calculate(self.x0, self.x1, self.x2)  # 計算適応度
```

•自分の最適な位置を更新する

```
def go_max(self, g):
    self.go(g)
    if self.y > self.p[3]:
        self.p = [self.x0, self.x1, self.x2, self.y]  # 自分の最適な位置を更新する

def go_min(self, g):
    self.go(g)
    if self.y < self.p[3]:
        self.p = [self.x0, self.x1, self.x2, self.y]  # 自分の最適な位置を更新する
```

3. 最大値、最小値をもとめる

•最大値

```
a = []
k = 0
best = []
for i in range(100):  # 初期粒子群を構造
    a.append(PSO())
    if a[k].y < a[i].y:
        k = i
best = a[k].p  # 最良解の値
for j in range(100):
    show = False
    for i in range(100):
        a[i].go_max(best)
        if a[i].y > best[3]:
```

```
                best = a[i].p
                show = True
        if show:
            print(best)
    print("最大値:", best[3], )
```

•最小値

```
for i in range(100):
    a.append(PSO())
    if a[k].y > a[i].y:
        k = i
best = a[k].p
for j in range(100):
    show = False
    for i in range(100):
        a[i].go_min(best)
        if a[i].y < best[3]:
            best = a[i].p
            show = True
    if show:
        print(best)
print("最小値:", best[3], )
```

4. 出力

```
[15, 0.7103407410240816, 15, 407.0379518000445]
[15, 0.7688586453619994, 15, 407.07086237716635]
最大値: 407.07086237716635
[1.061294111861832, 15, 0, -601.9924860637021]
[1.0477182169067847, 15, 0.0, -601.9954459435505]
[0.9673496896363303, 15, 0, -601.9978679144664]
[1.0088106098281133, 15.0, 0, -601.9998447463089]
最小値: -601.9998447463089

Process finished with exit code 0
```

5. 評価

（1）粒子群最適化法は不確定アルゴリズムの一種である。不確実性は自然界生物の生物機構を具現し、特定の問題を解決する上で決定性アルゴリズムより優れている。

（2）粒子群最適化法は確率型のグローバル最適化アルゴリズムです。利点はアルゴリ

ズムが大域最適解を解くより多くの機会を持つことである。

（3）問題そのものの厳密な数学的性質の最適化に依存しない。

（4）本質的な並列性を有する。内在的並列性と内在的並列性を含む。

（5）突出性がある。粒子群アルゴリズムの全目標の達成は，複数の知能バルクの個々の挙動の運動の間に突然現われる。

（6）自己組織と進化性と記憶機能を持ち、すべての粒子が優解に関する知識を保存している。

三．コード

1．遺伝的アルゴリズム

```
import numpy as np

N = 15 # 遺伝子長
Pc = 0.30 # 交叉確率
Pm = 0.05 # 突然変異確率
N_GENERTATIONS = 100 # 世代数
POP_SIZE = 200 # 群の行列
X1_BOUND = [0, 15] # x1 取値範囲
X2_BOUND = [0, 15] # x2 取値範囲
X3_BOUND = [0, 15] # x3 取値範囲


def F(x1, x2, x3):
    return 2 * x1 * x1 - 3 * x2 * x2 - 4 * x1 + 5 * x2 + x3;


def translateDNA(pop): # pop は群行列を表し、行はバイナリ符号化で表される DNA を表し、行列の行数は群の数である
    x1_pop = pop[:, :N]
    x2_pop = pop[:, N:-N]
    x3_pop = pop[:, -N:]
    x1 = x1_pop.dot(2 ** np.arange(N)[::-1]) / float(2 ** N - 1) * ¥
         (X1_BOUND[1] - X1_BOUND[0]) + X1_BOUND[0]
    x2 = x2_pop.dot(2 ** np.arange(N)[::-1]) / float(2 ** N - 1) * ¥
```

```
        (X2_BOUND[1] - X2_BOUND[0]) + X2_BOUND[0]
    x3 = x3_pop.dot(2 ** np.arange(N)[::-1]) / float(2 ** N - 1) * ¥
        (X3_BOUND[1] - X3_BOUND[0]) + X3_BOUND[0]
    return x1, x2, x3


def get_fitness(pop):
    x1, x2, x3 = translateDNA(pop)
    pred = F(x1, x2, x3)
    return pred - np.min(pred) + 1e-3# 最小の適応度を減算するのは、適応度に負が生じるのを防ぐためfitness の範囲は[0,np.max(pred)-np.min(pred)


def crossover(pop, rate=Pc):
    new_pop = []
    for father in pop:# 群れの中のすべての個体を巡り、その個体を父親とする
        child = father# 子供は父の遺伝子をもらう
        if np.random.rand() < rate:# 一定の確率で交差が発生する
            mother = pop[np.random.randint(POP_SIZE)]# もう一つの个体を母としてえらぶ
            cross_points = np.random.randint(low=0, high=N * 3)# ランダムに交差点を生成する
            child[cross_points:] = mother[cross_points:]# 子供は交差点の後ろにある母の遺伝子をもらう
        mutation(child)# 子供ごとに一定の確率で変異が起こる。
        new_pop.append(child)
    return new_pop


def mutation(child, rate=Pm):
    if np.random.rand() < rate:# rate の確率で変異する
        mutate_point = np.random.randint(0, N * 3)# ランダムに実数を生成し、遺伝子を変異させる位置を表す
        child[mutate_point] = child[mutate_point] ^ 1# 変異点のバイナリを反転しする


def select(pop, fitness):# 群を生成する
```

```
    idx = np.random.choice(np.arange(POP_SIZE), size=POP_SIZE,
                           replace=True, p=(fitness) / (fitness.sum()))
    return pop[idx]


def print_pre(pop):
    fitness = get_fitness(pop)
    max_fitness_index = np.argmax(fitness)
    fit=round(fitness[max_fitness_index], 2)
    print("max_fitness",fit)
    x1, x2, x3 = translateDNA(pop)
    y=F(x1[max_fitness_index], x2[max_fitness_index],
x3[max_fitness_index])
    y1=round(y,2)
    print("2*x1*x1-3*x2*x2-4*x1+5*x2+x3=",y1)


if __name__ == '__main__':
    pop = np.random.randint(2, size=(POP_SIZE, N * 3))# 群を生成する
    print(pop)
    for i in range(N_GENERTATIONS):# 群は反復して進化する N_GENERATIONS 代
        x1, x2, x3 = translateDNA(pop)
        pop = np.array(crossover(pop))# 交差変異を通して後代を生む
        fitness = get_fitness(pop)# 群の中の各個体を評価する
        pop = select(pop, fitness)# 新しいグループを作成する場合にする
        print("{:=^50}¥n{}番目".format("", i + 1))
        print_pre(pop)

def get_fitness(pop):
    x, y = translateDNA(pop)
    pred = F(x, y)
    return -(pred - np.max(pred))
```

2. 粒子群最適化法

```
import random
```

```
def calculate(x1, x2, x3):  # 適応度
    return 2 * x1 * x1 - 3 * x2 * x2 - 4 * x1 + 5 * x2 + x3

class PSO:
    def __init__(self, c1=2, c2=2, w=2):
        self.w = w  # 慣性定数
        self.v0 = 0  # 探索個体の初速度
        self.v1 = 0
        self.v2 = 0
        self.x0 = random.randint(0, 15)  # 探索個体の位置
        self.x1 = random.randint(0, 15)
        self.x2 = random.randint(0, 15)
        self.y = calculate(self.x0, self.x1, self.x2)
        self.c1 = c1  # 学習係数
        self.c2 = c2
        self.p = [self.x0, self.x1, self.x2, self.y]  # 探索個体の最良位置

    def go(self, g):  # 探索関数
        # 計算速度
        self.v0 = self.w * self.v0 + self.c1 * random.random() * (g[0] -
self.x0) + ¥
                  self.c2 * random.random() * (self.p[0] - self.x0)
        self.v1 = self.w * self.v1 + self.c1 * random.random() * (g[1] -
self.x1) + ¥
                  self.c2 * random.random() * (self.p[1] - self.x1)
        self.v2 = self.w * self.v2 + self.c1 * random.random() * (g[2] -
self.x2) + ¥
                  self.c2 * random.random() * (self.p[2] - self.x2)
        # 計算場所
        self.x0 = self.x0 + self.v0  # 限られた値の範囲
        if self.x0 > 15:
            self.x0 = 15
        elif self.x0 < 0:
            self.x0 = 0
        self.x1 = self.x1 + self.v1
        if self.x1 > 15:
            self.x1 = 15
        elif self.x1 < 0:
```

```
            self.x1 = 0
        self.x2 = self.x2 + self.v2
        if self.x2 > 15:
            self.x2 = 15
        elif self.x2 < 0:
            self.x2 = 0
        self.y = calculate(self.x0, self.x1, self.x2)  # 計算適応度

    def go_max(self, g):
        self.go(g)
        if self.y > self.p[3]:
            self.p = [self.x0, self.x1, self.x2, self.y]  # 自分の最適な位置を
更新する

    def go_min(self, g):
        self.go(g)
        if self.y < self.p[3]:
            self.p = [self.x0, self.x1, self.x2, self.y]  # 自分の最適な位置を
更新する

a = []
k = 0
best = []

for i in range(100):  # 初期粒子群を構造
    a.append(PSO())
    if a[k].y < a[i].y:
        k = i
best = a[k].p  # 最良解の値
for j in range(100):
    show = False
    for i in range(100):
        a[i].go_max(best)
        if a[i].y > best[3]:
            best = a[i].p
            show = True
    if show:
        print(best)
```

```
print("最大値:", best[3], )

a = []
k = 0
best = []

for i in range(100):
    a.append(PSO())
    if a[k].y > a[i].y:
        k = i
best = a[k].p
for j in range(100):
    show = False
    for i in range(100):
        a[i].go_min(best)
        if a[i].y < best[3]:
            best = a[i].p
            show = True
    if show:
        print(best)
print("最小値:", best[3], )
```