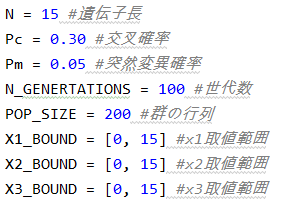
計算知能レポート

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

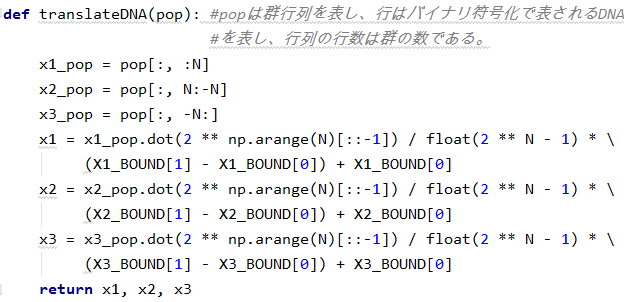


2.関数を実装する

•求めたいy

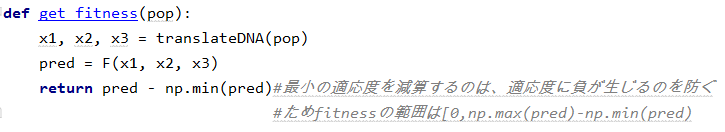


•復号プロセス

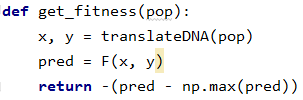


•最大値適応度計算関数

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。

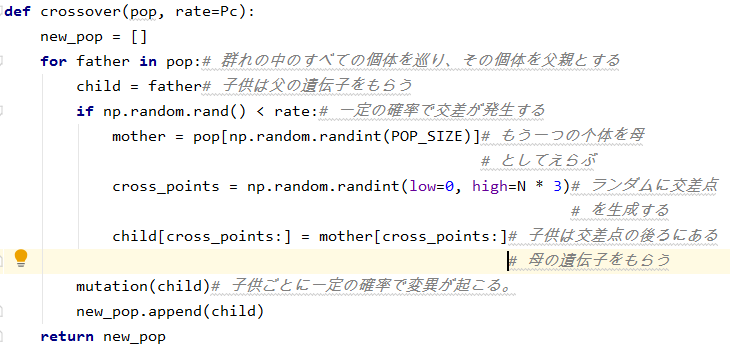


•最小値適応度計算関数



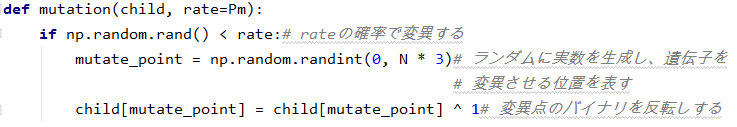
•一点交叉関数

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。



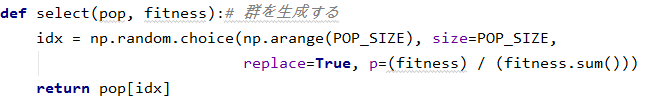
•突然変異関数

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。

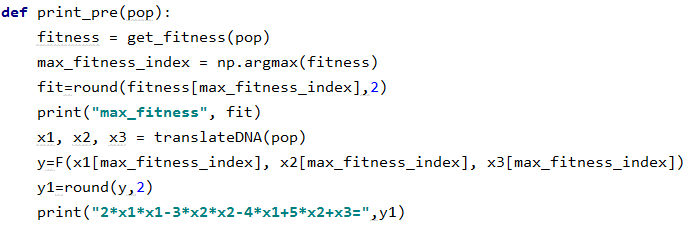


•群を生成する関数

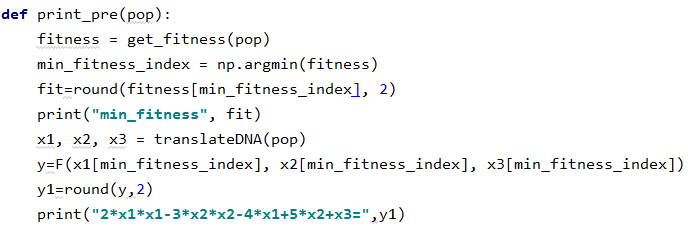
選択は新しい個体の適応度によって行われるが、同時に完全に適応度の高低をガイドにするという意味ではなく、単純に適応度の高い個体を選ぶとアルゴリズムが大域最適解ではなく局所最適解に急速に収束する可能性があるからである。遺伝的アルゴリズムは原則によって適応度が高いほど、選択される機会が高く、適応度が低いほど、選択される機会が低い。



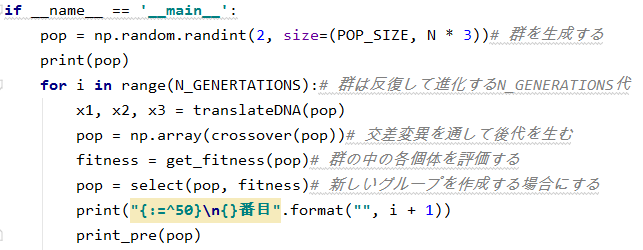
•最大値計算関数



•最小値計算関数

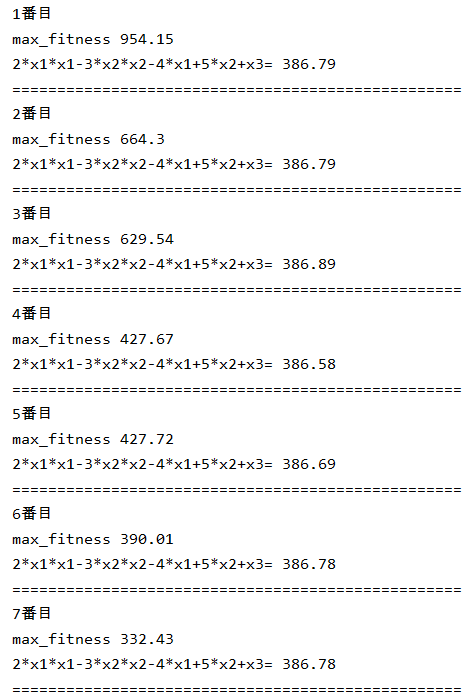


•main関数

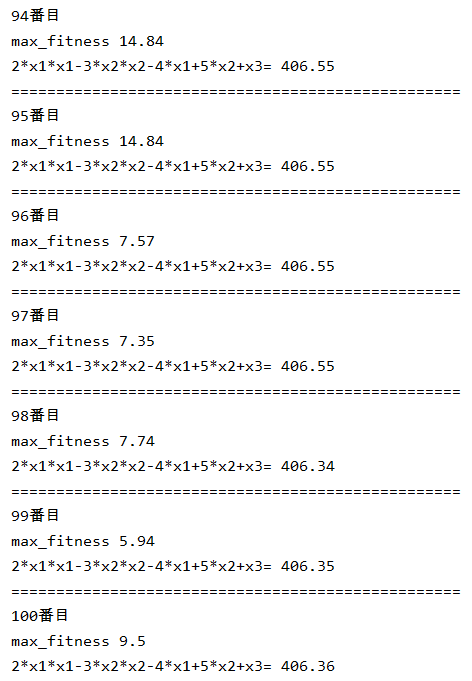


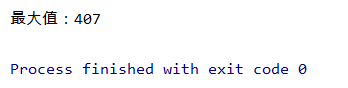
3.出力

•最大值

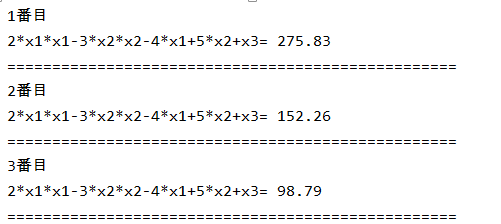


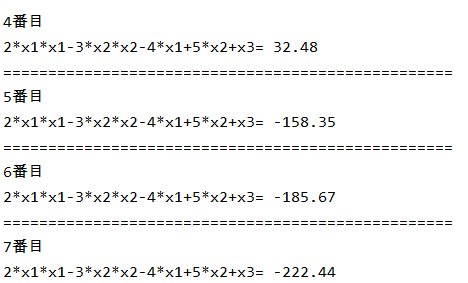
ここは省略します。



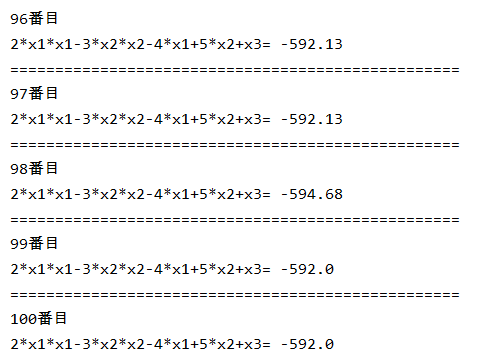


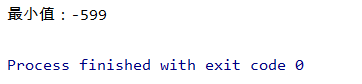
•最小值





ここは省略します。



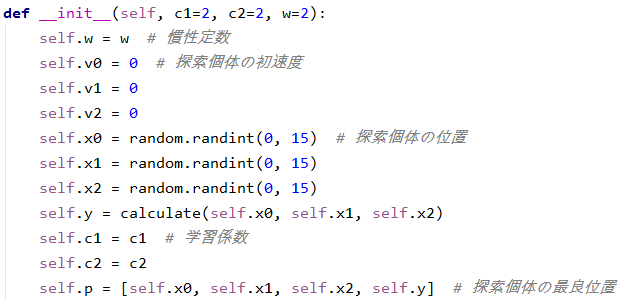


4.評価

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。

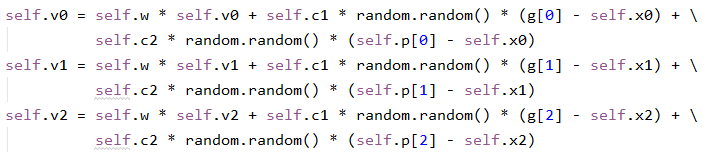
二．粒子群最適化法

1.変数を宣言する

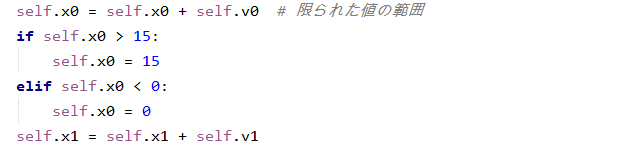


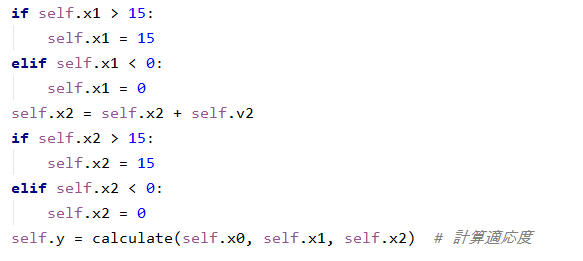
2.探索関数

•速度をもとめる

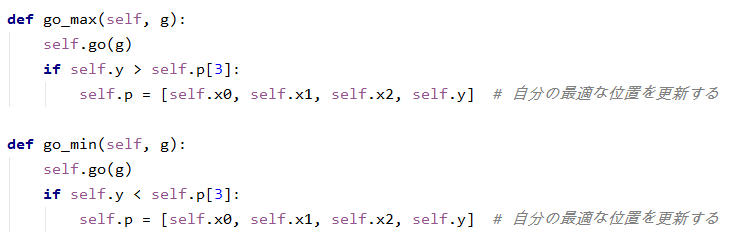


•場所をもとめる



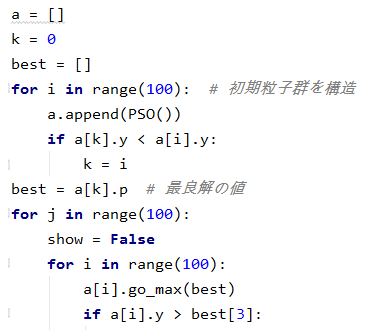


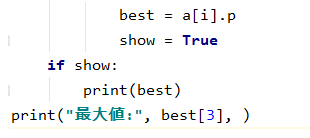
•自分の最適な位置を更新する



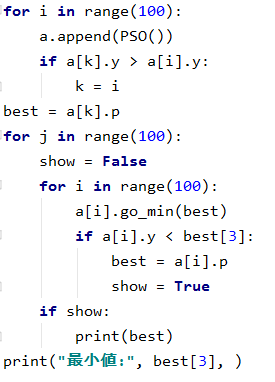
3.最大值、最小值をもとめる

•最大值

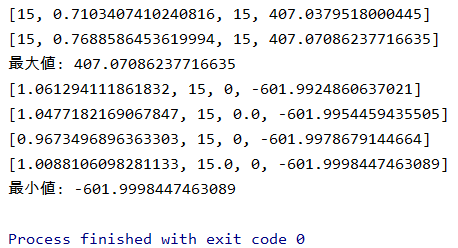




•最小值



4.出力



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], )