**#遺伝的アルゴリズム ソースコード1**

**import** **matplotlib.pyplot** **as** **plt**

**import** **numpy** **as** **np**

N = 4 *#遺伝子長*

M = 100 *#個体数*

Pc = 0.30 *#交叉確率*

Pm = 0.10 *#突然変異確率*

T = 100 *#世代数*

G\_max = [] *#最良解集合*

*#適応度計算*

**def** evaluation(individuals):

x1\_binary = individuals[:, 0:4]

x2\_binary = individuals[:, 4:8]

x3\_binary = individuals[:, 8:12]

*#2進数から10進数*

x1 = x1\_binary.dot(2 \*\* np.arange(N))

x2 = x2\_binary.dot(2 \*\* np.arange(N))

x3 = x3\_binary.dot(2 \*\* np.arange(N))

*# 目的関数*

pred = F(x1, x2, x3)

*# 最大適応度集合*

G\_max.append(pred.max())

**return** (pred - np.min(pred))

*#目的関数*

**def** F(x1, x2, x3):

**return** 2\*x1\*\*2-3\*x2\*\*2-4\*x1+5\*x2+x3

**#遺伝的アルゴリズム ソースコード2**

*#交叉*

**def** crossover(individuals,Pc):

new\_individuals = []

*#個体をランダムにペアリング(親個体ia, ibを選ぶ)*

**for** ia **in** individuals:*#親iaを決定*

child = ia

**if** np.random.rand() < Pc:*#交叉*

ib = individuals[np.random.randint(M)]*#親ibを決定*

cross\_points = np.random.randint(low=0, high=N\*3)*#交叉点*

child[cross\_points:] = ib[cross\_points:]

mutation(child)*#突然変異*

new\_individuals.append(child)

**return** new\_individuals

*#突然変異***def** mutation(child):

**if** np.random.rand() < Pm:

mutate\_point = np.random.randint(0, N)

child[mutate\_point] = child[mutate\_point]^1

*#ルーレット選択***def** roulette\_selection(individuals, fitness):

*# fitness = np.square(fitness)*

idx = np.random.choice(np.arange(M), size=M, replace=**True**,p=(fitness)/(fitness.sum())) *#ルーレット・ホイールに従って次世代を決定*

**return** individuals[idx]

**#遺伝的アルゴリズム ソースコード2**

*#交叉*

**def** crossover(individuals,Pc):

new\_individuals = []

*#個体をランダムにペアリング(親個体ia, ibを選ぶ)*

**for** ia **in** individuals:*#親iaを決定*

child = ia

**if** np.random.rand() < Pc:*#交叉*

ib = individuals[np.random.randint(M)]*#親ibを決定*

cross\_points = np.random.randint(low=0, high=N\*3)*#交叉点*

child[cross\_points:] = ib[cross\_points:]

mutation(child)*#突然変異*

new\_individuals.append(child)

**return** new\_individuals

*#突然変異*

**def** mutation(child):

**if** np.random.rand() < Pm:

mutate\_point = np.random.randint(0, N)

child[mutate\_point] = child[mutate\_point]^1

*#ルーレット選択*

**def** roulette\_selection(individuals, fitness):

*# fitness = np.square(fitness)*

idx = np.random.choice(np.arange(M), size=M, replace=**True**,p=(fitness)/(fitness.sum())) *#ルーレット・ホイールに従って次世代を決定*

**return** individuals[idx]

**#遺伝的アルゴリズム ソースコード3**

**if** \_\_name\_\_ == "\_\_main\_\_":

*#0世代目生成*

individuals = np.random.randint(2, size=(M, N\*3))

*# 適応度計算*

fitness = evaluation(individuals)

*#1～T世代生成*

**for** \_ **in** range(T):

*#交叉,突然変異*

individuals = np.array(crossover(individuals, Pc))

*#適応度値計算*

fitness = evaluation(individuals)

*#ルーレット選択*

individuals = roulette\_selection(individuals, fitness)

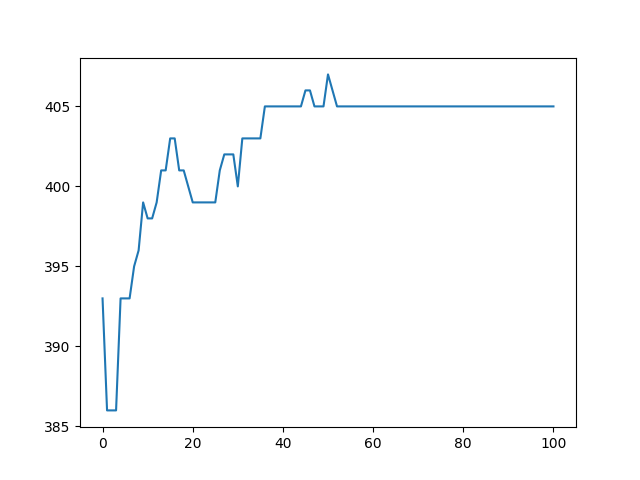
*#結果を示す*

plt.plot(G\_max)

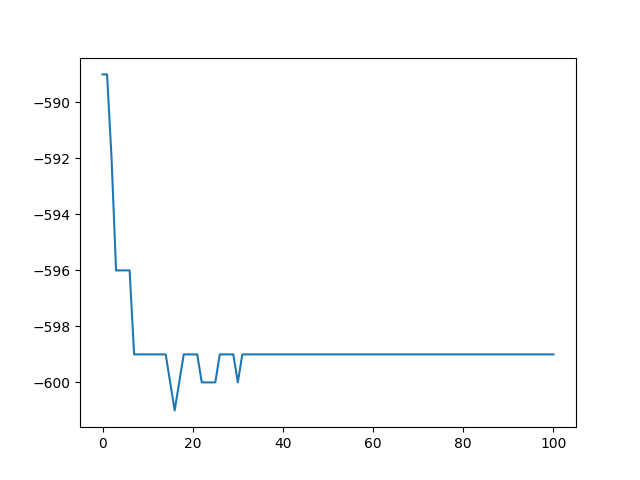
plt.show()

交叉確率:0.30

突然変異確率:0.10



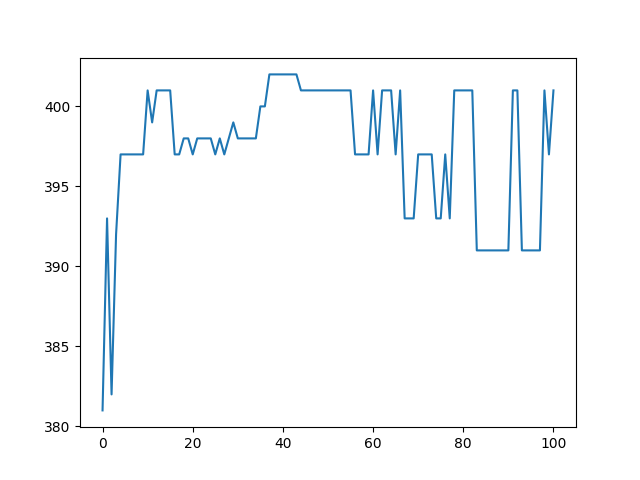
最良解（最大化）



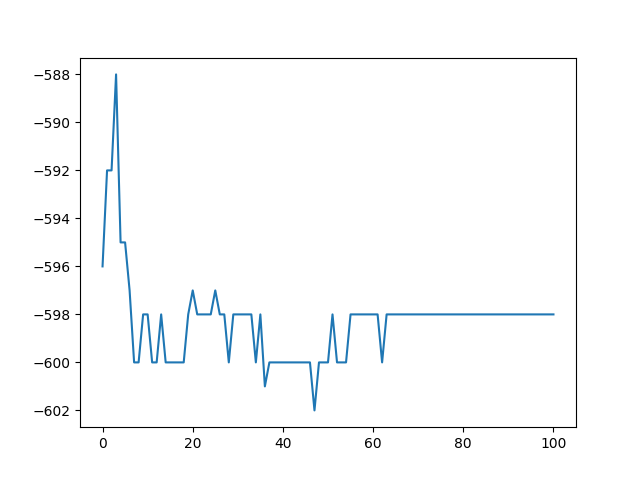
最良解（最小化）

交叉確率:0.30

突然変異確率:0.50



最良解（最大化）



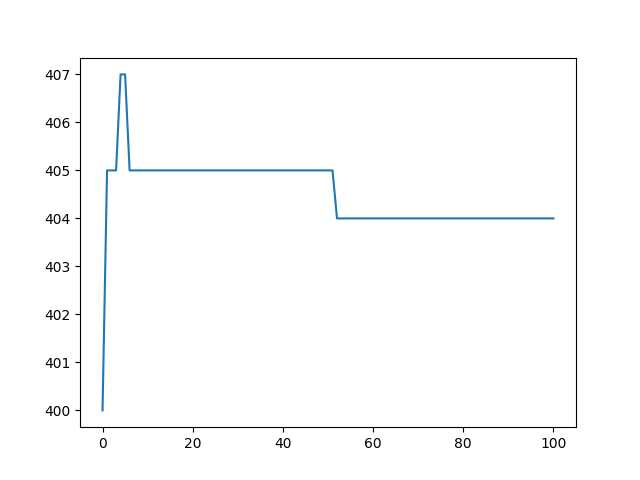
最良解（最小化）

**突然変異の確率は0.1%～1%、高くても数%である。確率が低すぎると局所的最適解に陥りやすくなり、高すぎるとランダム探索に近づいてしまう（解が収束しにくくなる）。**

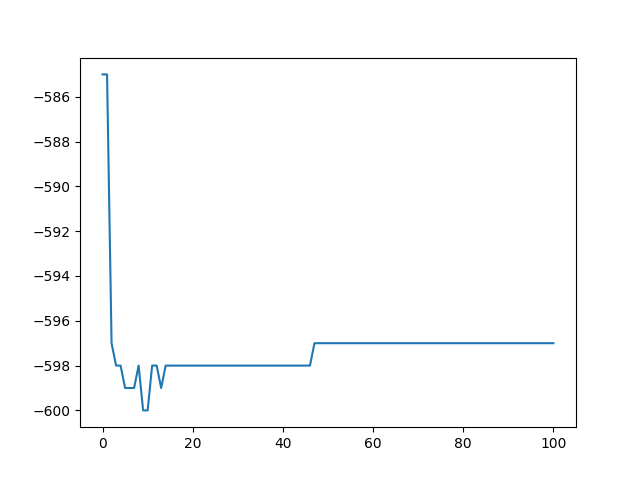
選択における確率を適応度に比例させるのではなく、適応度の2乗に比例させると

交叉確率:0.30

突然変異確率:0.10



最良解（最大化）

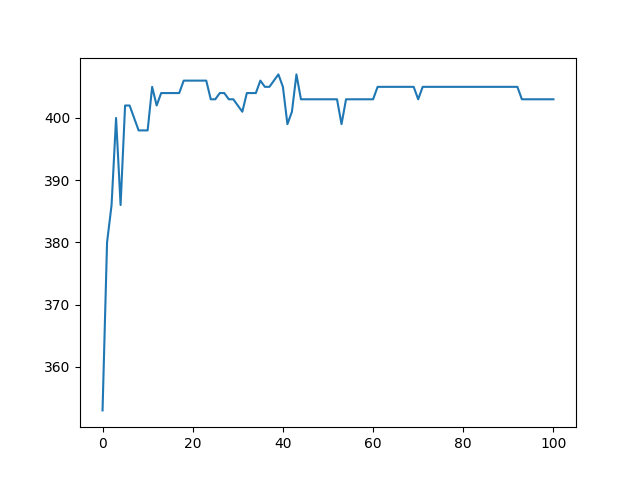


最良解（最小化）

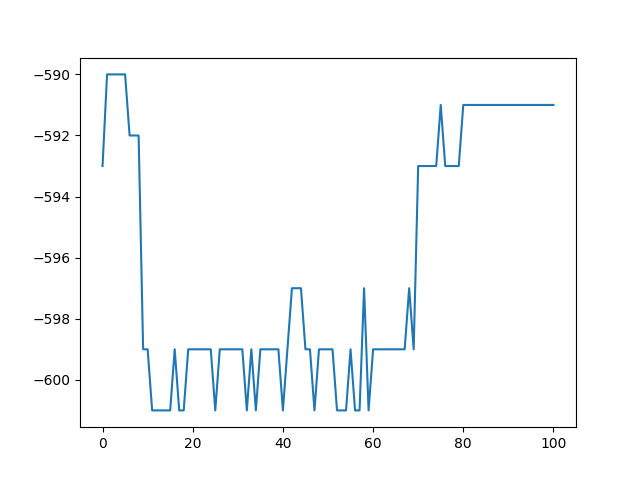
選択における確率を適応度に比例させるのではなく、適応度の平方根に比例させると

交叉確率:0.30

突然変異確率:0.10



最良解（最大化）



最良解（最小化）

**#粒子群最適化法 ソースコード1**

**import** **numpy** **as** **npimport** **matplotlib.pyplot** **as** **plt**

M = 100 *#個体数*

T = 100 *#世代数*

C1 = 2 *#学習係数c1*

C2 = 2 *#学習係数c2*

W = 0.6 *#慣性定数*

X\_MIN = 0 *#最小値*

X\_MAX = 15 *#最大値*

G\_max = [] *#最良解集合*

*#目的関数*

**def** F(x1, x2, x3):

**return** 2\*x1\*\*2-3\*x2\*\*2-4\*x1+5\*x2+x3

**def** clamp(n, minn, maxn):

n = np.maximum(n, minn)

n = np.minimum(n, maxn)

**return** n

**class** **PSO**(object):

**def** \_\_init\_\_(self, individual\_size=M, max\_steps=T):

self.individual\_size = individual\_size *# 粒子群数*

self.dim = 3 *# 空間次元*

self.max\_steps = max\_steps *# 世代数*

self.x\_bound = [X\_MIN, X\_MAX] *# 解集団の範囲*

self.x = np.random.randint(self.x\_bound[0],self.x\_bound[1]+1, size=(M, self.dim))

self.v = np.random.rand(self.individual\_size, self.dim) *# 探索個体の速度*

fitness = self.calculate\_fitness(self.x)

self.p = self.x *# 探索個体の最良位置*

self.pg = self.x[np.argmax(fitness)] *# 個体群全体での最良位置*

self.individual\_best\_fitness = fitness *# 個体の最良適応度*

self.global\_best\_fitness = np.max(fitness) *# 個体群全体での最良適応度*

G\_max.append(self.global\_best\_fitness)

**#粒子群最適化法 ソースコード2**

**def** calculate\_fitness(self, x):

x1 = x[:,0]

x2 = x[:,1]

x3 = x[:,2]

pred = F(x1, x2, x3)

**return** pred

**def** evolve(self):

**for** \_ **in** range(self.max\_steps):

r1 = np.random.rand(self.individual\_size, self.dim)

r2 = np.random.rand(self.individual\_size, self.dim)

*# 学習係数と速度の更新*

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

self.x = clamp(self.v + self.x, minn, maxn)

fitness = self.calculate\_fitness(self.x)

*# 個体の更新*

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]

*# 個体群全体での最良適応度と最良位置の更新*

**if** np.max(fitness) > self.global\_best\_fitness:

self.pg = self.x[np.argmax(fitness)]

self.global\_best\_fitness = np.max(fitness)

print('best fitness: **%.5f**, mean fitness: **%.5f**' % (self.global\_best\_fitness, np.mean(fitness)))

G\_max.append(np.max(fitness))

pso = PSO()

pso.evolve()

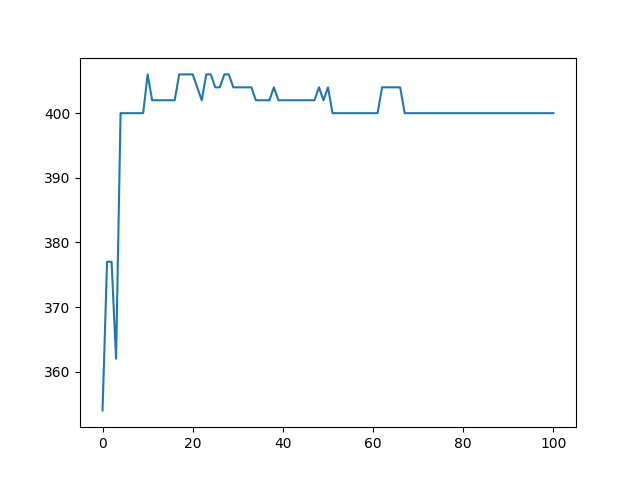
plt.plot(G\_max)

plt.show()

学習係数C1:2

学習係数C2:2

慣性定数W:0.6



最良解

**W:**

**慣性定数。多くの場合 1 より若干小さい値が最適である。**

**C1&C2:**

**群のうちで良い位置に向かう粒子の割合。1 に近い値が多くの場合最適である。**