**计算智能 作业4**

2021年11月09日

**要求：**

1. 在文档中说明解题思路、方法实现、求解结果等，必要时需要对结果进行分析和讨论。
2. 将源代码作为附录粘贴于作业文档末尾，并同时作为提交的附件与作业文档一起放在压缩包内备查。
3. 独立完成，严禁抄袭。

**内容：**

1. 编写程序实现论文“一种多尺度协同变异的粒子群优化算法”（哈尔滨工程大学陶新民等,《软件学报》，2012年07期）的部分测试实验。

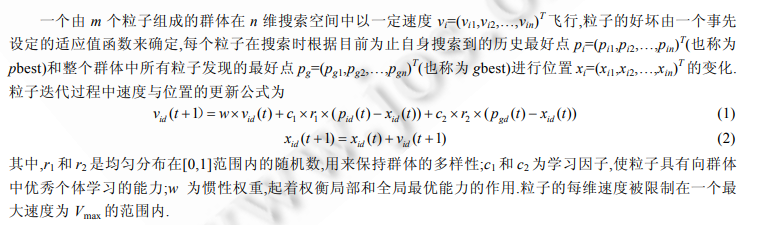
说明：

1. 至少编程实现传统PSO算法和论文所提出的MAEPSO算法。
2. 至少测试论文中的三个Benchmark函数。

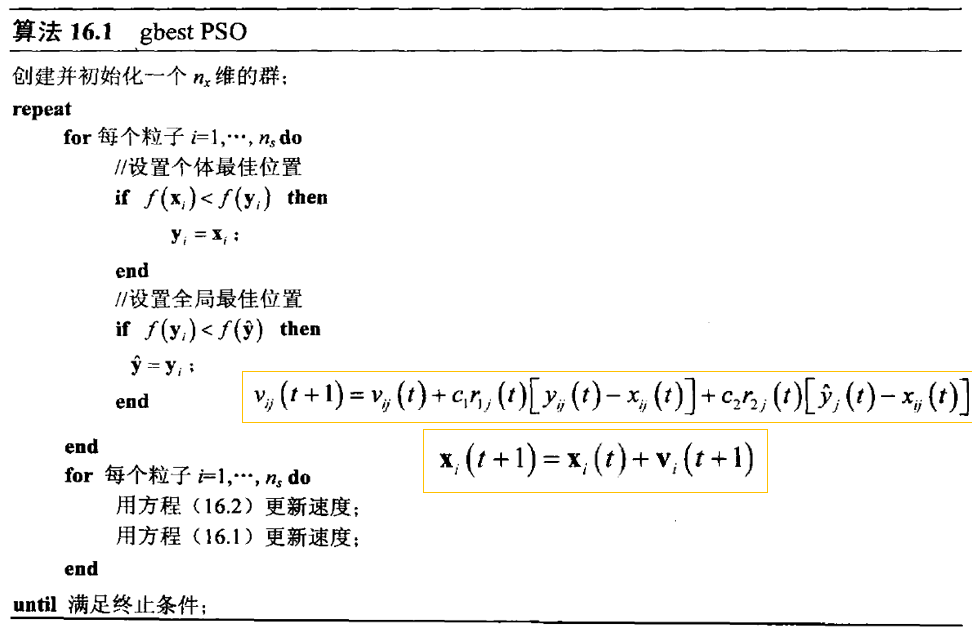
3) 编程语言不限（推荐C/C++，需提交源代码和运行结果截图）。

解析：

传统PSO算法：



根据PPT的算法：



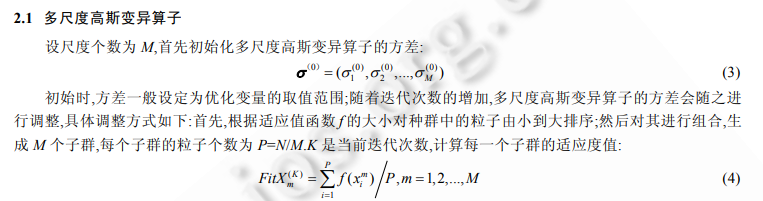
更新速度算法：

# 按照公式更新速度  
def update\_speed(self, part):  
 for i in range(self.dim):  
 speed\_value = self.W \* part.get\_speed()[i] + self.C1 \* random.random() \* (part.get\_pbest()[i] - part.get\_pos()[i]) \  
 + self.C2 \* random.random() \* (self.get\_gbest()[i] - part.get\_pos()[i])  
 if speed\_value > self.maxSpeed:  
 speed\_value = self.maxSpeed  
 elif speed\_value < -self.maxSpeed:  
 speed\_value = -self.maxSpeed  
 part.set\_speed(i, speed\_value)

在这里我们使用gbest PSO算法，每次更新的同时，根据fitness同时更新gbest和gbest

def update\_pos(self, part):  
 for i in range(self.dim):  
 pos\_value = part.get\_pos()[i] + part.get\_speed()[i]  
 part.set\_pos(i, pos\_value)  
 value = fitnessFunction(part.get\_pos())  
 if value < part.get\_fitness\_value():  
 part.set\_fitness\_value(value)  
 for i in range(self.dim):  
 part.set\_pbest(i, part.get\_pos()[i])  
 if value < self.get\_bestFitnessValue():  
 self.set\_bestFitnessValue(value)  
 for i in range(self.dim):  
 self.set\_gbest(i, part.get\_pos()[i])

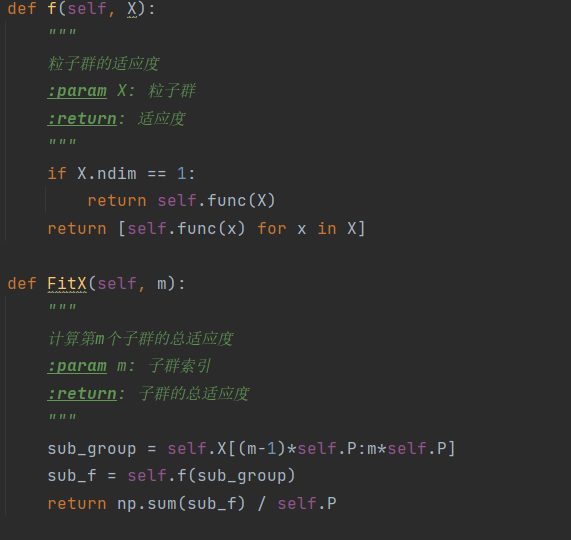
MAEPSO算法：

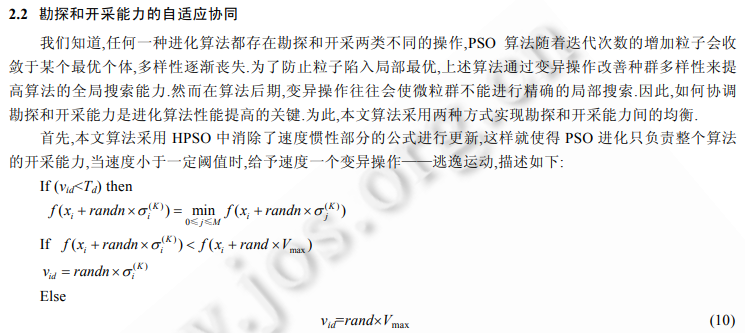


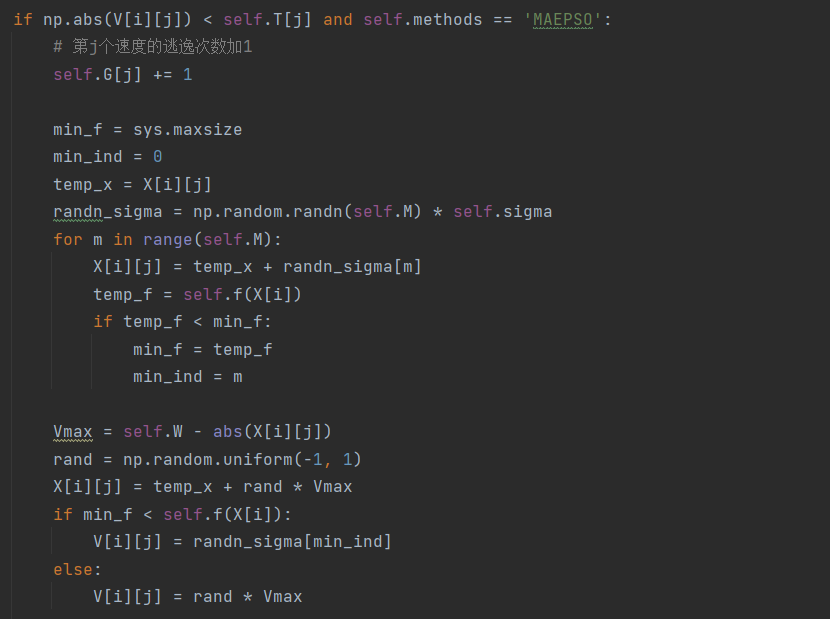
这里的关键之一是多尺度，即多个子群体。

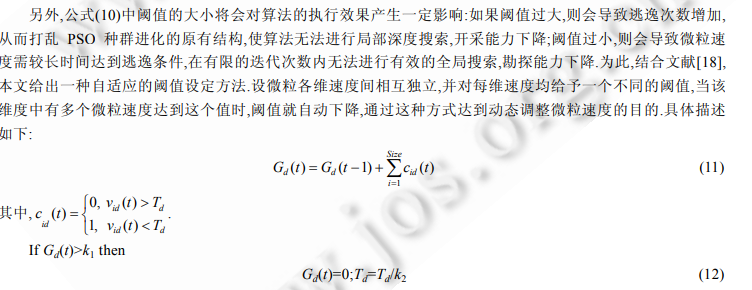


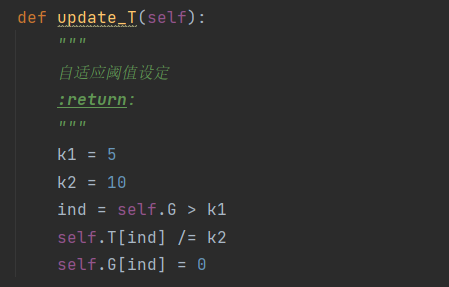
这里是适应度的计算









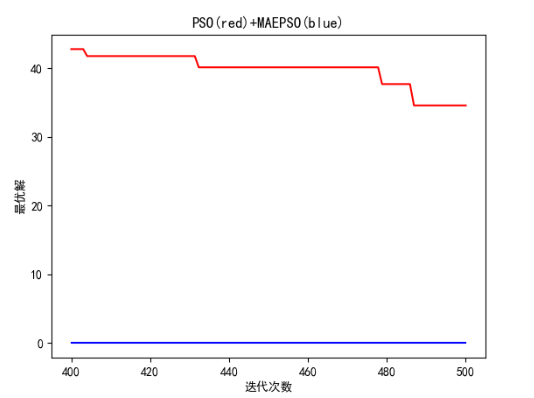


通过这些关键函数，将这两个算法实现之后，我们做以下对比：

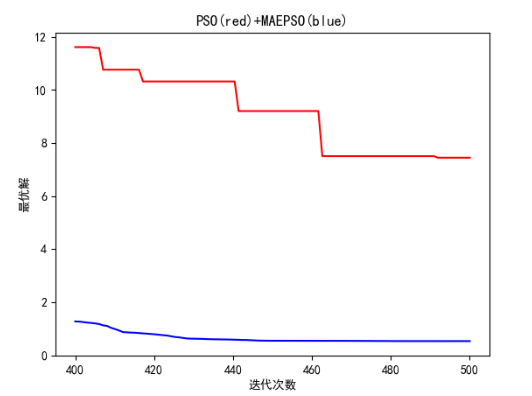
在这里我们设定迭代范围为500、个体维度为30维、个体数目为20，

为了强调对比结果，我们只看400-500迭代时候的最优解

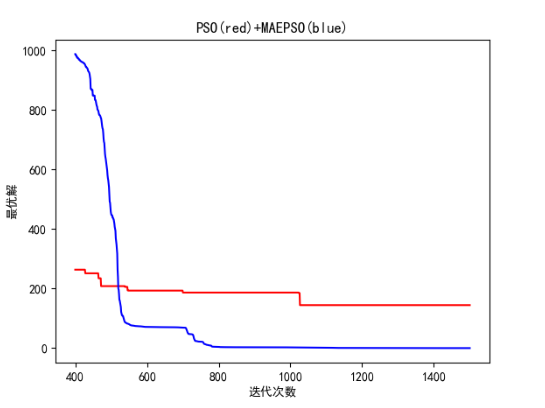
Tablet：



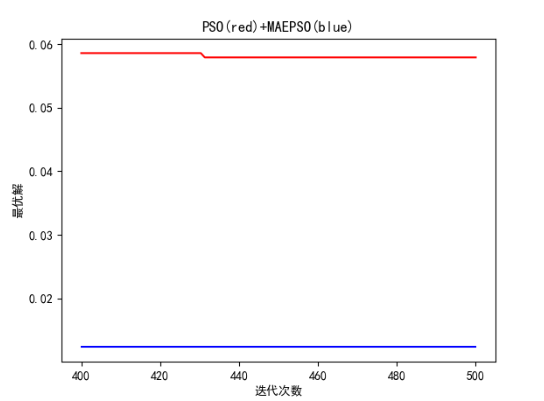
Quadratic：



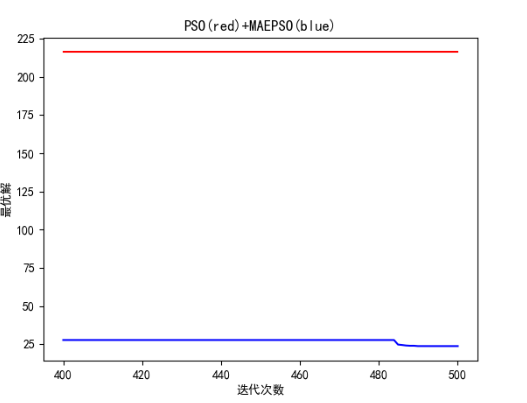
Rosenbrock：



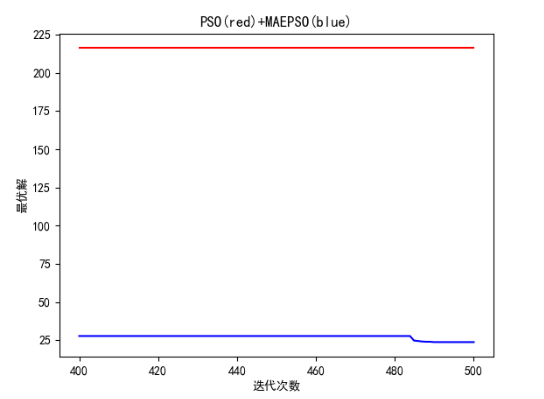
Griewank：



Rastrigin：



Schaffer：



## 代码

import math  
  
  
def Tablet(X):  
 sum = 0  
 for i in range(1, len(X)):  
 sum = sum + X[i] \*\* 2  
 return sum + (X[0] \*\* 2) \* 100000  
  
  
def Quadratic(X):  
 sum = 0  
 for i in range(0, len(X)):  
 tmp = 0  
 for j in range(0, i + 1):  
 tmp += X[j]  
 sum += tmp \*\* 2  
 return sum  
  
  
def Rosenbrock(X):  
 sum = 0  
 for i in range(0, len(X) - 1):  
 sum += 100 \* (X[i + 1] - X[i] \* X[i]) \*\* 2 + (X[i] - 1) \*\* 2  
 return sum  
  
  
def Griewank(X):  
 sum1 = 0  
 sum2 = 1  
 for i in range(0, len(X)):  
 sum1 += X[i] \* X[i] / 4000  
 sum2 \*= math.cos(X[i] / math.sqrt(i + 1))  
 return sum1 - sum2 + 1  
  
  
def Rastrigin(X):  
 sum = 0  
 for i in range(0, len(X)):  
 sum += (X[i] \*\* 2 - 10 \* math.cos(2 \* math.pi \* X[i]) + 10)  
 return sum  
  
  
def Schaffer(X):  
 sum = 0  
 for i in range(0, len(X) - 1):  
 si = pow(X[i] \*\* 2 + X[i + 1] \*\* 2, 1 / 4)  
 sum += si \* (math.sin(50 \* pow(si, 1 / 10)) + 1.0)  
 return sum

import sys  
import numpy as np  
import matplotlib.pyplot as plt  
from benchmark\_function import \*  
  
class MAEPSO(object):  
 *"""  
 MAEPSO算法  
 陶新民, 刘福荣, 刘玉,等. 一种多尺度协同变异的粒子群优化算法[J]. 软件学报, 2012, 23(7):1805-1815.  
 """* def \_\_init\_\_(self, func=lambda x:np.sum(np.square(x), axis=1), methods='MAEPSO'):  
 *"""  
 算法参数设置* ***:param*** *func: 适应度函数  
 """* self.size = 20 # 粒子群数量  
 self.M = 5 # 尺度个数  
 self.dim = 30 # 函数维度  
 self.W = 100 # 变量空间的宽度  
 self.c1 = 1.2 # 个体最优值的学习因子  
 self.c2 = 1.2 # 全局最优值的学习因子  
 self.w\_max = 0.8 # 最大惯性因子  
 self.w\_min = 0.4 # 最小惯性因子  
 self.w = self.w\_max # 惯性因子  
 self.P = (int)(self.size / self.M) # 每个子群的粒子个数  
 self.func = func # 适应度函数  
 self.methods = methods # 粒子群算法  
 self.bestFitnessList = []  
 self.\_init\_PSO()  
  
 def \_init\_PSO(self):  
 *"""  
 位置、速度等变量初始化* ***:return****:  
 """* self.X = np.random.uniform(-self.W, self.W, (self.size, self.dim)) # 初始化粒子位置  
 self.pb = np.copy(self.X) # 个体最优位置  
 self.gb = self.X[np.argmin(self.f(self.X))] # 全局最优位置  
 self.V = np.random.uniform(-1, 1, (self.size, self.dim)) # 初始化粒子速度  
 self.T = np.ones(self.dim) \* 0.5 # 速度的阈值  
 self.G = np.zeros(self.dim) # 速度的逃逸次数  
 self.sigma = np.ones(self.M) \* 2 \* self.W # M个尺度的高斯变异算子方差  
  
 def f(self, X):  
 *"""  
 粒子群的适应度* ***:param*** *X: 粒子群* ***:return****: 适应度  
 """* if X.ndim == 1:  
 return self.func(X)  
 return [self.func(x) for x in X]  
  
 def FitX(self, m):  
 *"""  
 计算第m个子群的总适应度* ***:param*** *m: 子群索引* ***:return****: 子群的总适应度  
 """* sub\_group = self.X[(m-1)\*self.P:m\*self.P]  
 sub\_f = self.f(sub\_group)  
 return np.sum(sub\_f) / self.P  
  
 def update\_sigma(self):  
 *"""  
 更新所有高斯变异算子的方差* ***:return****:  
 """* FitXs = [self.FitX(m+1) for m in range(self.M)]  
 max\_FitX = np.max(FitXs)  
 min\_Fitx = np.min(FitXs)  
 total\_FitX = np.sum(FitXs)  
  
 for i in range(self.M):  
 self.sigma[i] \*= np.exp((self.M \* FitXs[i] - total\_FitX) / (max\_FitX - min\_Fitx + pow(10, -10)))  
  
 while self.sigma[i] > self.W / 4:  
 self.sigma[i] -= self.W / 4  
  
 def update\_T(self):  
 *"""  
 自适应阈值设定* ***:return****:  
 """* k1 = 5  
 k2 = 10  
 ind = self.G > k1  
 self.T[ind] /= k2  
 self.G[ind] = 0  
  
 def update\_VP(self, methods='MASPSO'):  
 *"""  
 更新粒子群的速度和位置* ***:return****:  
 """* V = self.V  
 X = self.X  
  
 for i in range(self.size):  
 for j in range(self.dim):  
  
 # 速度更新公式  
 V[i][j] = self.w \* V[i][j] + \  
 self.c1 \* np.random.rand() \* (self.pb[i][j] - X[i][j]) + \  
 self.c2 \* np.random.rand() \* (self.gb[j] - X[i][j])  
  
 # 判断是否需要逃逸, 若满足逃逸条件，则进行逃逸  
 if np.abs(V[i][j]) < self.T[j] and self.methods == 'MAEPSO':  
 # 第j个速度的逃逸次数加1  
 self.G[j] += 1  
  
 min\_f = sys.maxsize  
 min\_ind = 0  
 temp\_x = X[i][j]  
 randn\_sigma = np.random.randn(self.M) \* self.sigma  
 for m in range(self.M):  
 X[i][j] = temp\_x + randn\_sigma[m]  
 temp\_f = self.f(X[i])  
 if temp\_f < min\_f:  
 min\_f = temp\_f  
 min\_ind = m  
  
 Vmax = self.W - abs(X[i][j])  
 rand = np.random.uniform(-1, 1)  
 X[i][j] = temp\_x + rand \* Vmax  
 if min\_f < self.f(X[i]):  
 V[i][j] = randn\_sigma[min\_ind]  
 else:  
 V[i][j] = rand \* Vmax  
  
 X[i][j] = temp\_x  
  
 # 更新粒子的位置  
 X[i][j] += V[i][j]  
 # 更新个体经历过的最优位置  
 if self.f(self.pb[i]) > self.f(X[i]):  
 self.pb[i] = X[i]  
 # 更新全局最优位置  
 if self.f(self.gb) > self.f(self.pb[i]):  
 self.gb = self.pb[i]  
  
 self.V = V  
 self.X = X  
  
 def evolve(self, steps=50, trials=1):  
 *"""  
 粒子群进化过程* ***:param*** *steps: 迭代次数* ***:param*** *trials: 实验次数* ***:return****:  
 """* w\_max = self.w\_max  
 w\_min = self.w\_min  
 results = np.array([])  
 print(self.func.\_\_name\_\_)  
 for t in range(trials):  
 self.\_init\_PSO()  
 plt.figure()  
 fitness\_iter = np.array([])  
 for s in range(steps):  
 self.update\_VP()  
 self.update\_sigma()  
 self.update\_T()  
 # HPSO论文(3)，消除速度惯性部分的公式  
 self.w = w\_max - (w\_max - w\_min) \* s / steps  
  
 self.bestFitnessList.append(self.f(self.gb))  
 if self.f(self.gb) == 0:  
 print("finished")  
 break  
 fitness\_iter = np.append(fitness\_iter, self.f(self.gb))  
  
 res = self.f(self.gb)  
 results = np.append(results, res)  
 print("trial {}: final fitness: {}".format(t+1, res))  
  
 results = np.array(results)  
 print("min fitness: {}".format(np.min(results)))  
 print("median fitness: {}".format(np.median(results)))  
 print("varience fitness: {}".format(np.var(results)))  
 print("mean fitness: {}".format(np.mean(results)))  
 print("max fitness: {}".format(np.max(results)))  
 print("")  
 return self.bestFitnessList

import random  
  
import matplotlib.pyplot as plt  
  
from MAEPSO import \*  
plt.rcParams['font.sans-serif'] = ['SimHei']  
  
  
iter\_num = 1500 # 迭代次数  
  
  
class Particle:  
 def \_\_init\_\_(self, Vmax, maxSpeed, dim, func=lambda x:np.sum(np.square(x), axis=1)):  
 self.pos = [random.uniform(-Vmax, Vmax) for i in range(dim)] # 粒子的位置  
 self.speed = [random.uniform(-maxSpeed, maxSpeed) for i in range(dim)] # 粒子的速度  
 self.bestPos = [0.0 for i in range(dim)] # 粒子最好的位置  
 self.func = func  
 self.fitnessValue = self.func(self.pos) # 适应度函数值  
  
 def set\_pos(self, i, value):  
 self.pos[i] = value  
  
 def get\_pos(self):  
 return self.pos  
  
 def set\_best\_pos(self, i, value):  
 self.bestPos[i] = value  
  
 def get\_best\_pos(self):  
 return self.bestPos  
  
 def set\_speed(self, i, value):  
 self.speed[i] = value  
  
 def get\_speed(self):  
 return self.speed  
  
 def set\_fitness\_value(self, value):  
 self.fitnessValue = value  
  
 def get\_fitness\_value(self):  
 return self.fitnessValue  
  
  
class PSO:  
 def \_\_init\_\_(self, iter\_num, func=lambda x:np.sum(np.square(x), axis=1), bestFitnessValue=float('Inf')):  
 self.C1 = 1.2  
 self.C2 = 1.2  
 self.W = 1  
 self.dim = 30 # 粒子的维度  
 self.n = 20 # 粒子个数  
 self.iter\_num = iter\_num # 迭代次数  
 self.Vmax = 10  
 self.maxSpeed = 0.5 # 粒子最大速度  
 self.bestFitnessValue = bestFitnessValue  
 self.bestPosition = [0.0 for i in range(self.dim)] # 种群最优位置  
 self.bestFitnessList = [] # 每次迭代最优适应值  
 self.func = func # 适应度函数  
  
 # 对种群进行初始化  
 self.ParticleList = [Particle(self.Vmax, self.maxSpeed, self.dim, self.func) for i in range(self.n)]  
  
 def set\_bestFitnessValue(self, value):  
 self.bestFitnessValue = value  
  
 def get\_bestFitnessValue(self):  
 return self.bestFitnessValue  
  
 def set\_bestPosition(self, i, value):  
 self.bestPosition[i] = value  
  
 def get\_bestPosition(self):  
 return self.bestPosition  
  
 # 按照公式更新速度  
 def update\_speed(self, part):  
 for i in range(self.dim):  
 speed\_value = self.W \* part.get\_speed()[i] + self.C1 \* random.random() \* (part.get\_best\_pos()[i] - part.get\_pos()[i]) \  
 + self.C2 \* random.random() \* (self.get\_bestPosition()[i] - part.get\_pos()[i])  
 if speed\_value > self.maxSpeed:  
 speed\_value = self.maxSpeed  
 elif speed\_value < -self.maxSpeed:  
 speed\_value = -self.maxSpeed  
 part.set\_speed(i, speed\_value)  
  
 # 按照公式更新位置  
 def update\_pos(self, part):  
 for i in range(self.dim):  
 pos\_value = part.get\_pos()[i] + part.get\_speed()[i]  
 part.set\_pos(i, pos\_value)  
 value = self.func(part.get\_pos())  
 if value < part.get\_fitness\_value():  
 part.set\_fitness\_value(value)  
 for i in range(self.dim):  
 part.set\_best\_pos(i, part.get\_pos()[i])  
 if value < self.get\_bestFitnessValue():  
 self.set\_bestFitnessValue(value)  
 for i in range(self.dim):  
 self.set\_bestPosition(i, part.get\_pos()[i])  
  
 def update(self):  
 for i in range(self.iter\_num):  
 for part in self.ParticleList:  
 self.update\_speed(part) # 更新速度  
 self.update\_pos(part) # 更新位置  
 self.bestFitnessList.append(self.get\_bestFitnessValue()) # 每次迭代完把当前的最好适应度保存  
 return self.bestFitnessList  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 pso1 = PSO(iter\_num, func=Rosenbrock)  
 bestFitnessList1 = pso1.update()  
 pso2 = MAEPSO(func=Rosenbrock, methods='MAEPSO')  
 bestFitnessList2 = pso2.evolve(iter\_num, 1)  
 x = np.linspace(400, iter\_num, iter\_num-400)  
 plt.figure()  
 plt.plot(x, bestFitnessList1[400:], c="r")  
 plt.plot(x, bestFitnessList2[400:], c="b")  
 plt.title("PSO(red)+MAEPSO(blue)")  
 plt.xlabel("迭代次数")  
 plt.ylabel("最优解")  
 plt.show()