粒子群算法代码分析

# 1 功能描述

粒子群优化算法源于对鸟群捕食的行为研究，是一种进化计算技术。粒子群优化算法的基本思想是通过群体中个体之间的协作和信息共享来寻找最优解。

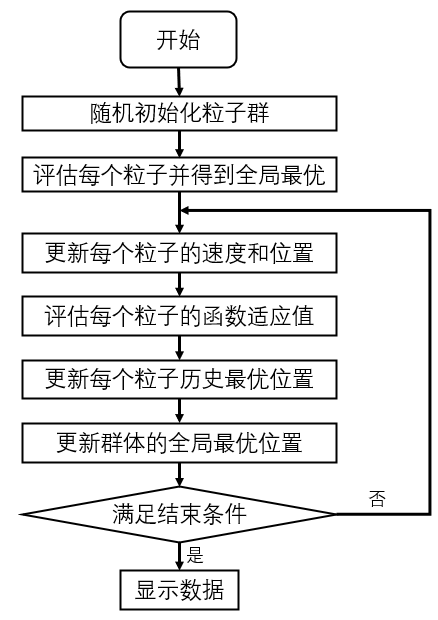
粒子群算法通过设计一种无质量的粒子来模拟鸟群中的鸟，粒子仅具有两个属性：速度和位置，速度代表移动的快慢，位置代表移动的方向。每个粒子在搜索空间中单独的搜寻最优解，并将其记为当前个体极值，并将个体极值与整个粒子群里的其他粒子共享，找到最优的那个个体极值作为整个粒子群的当前全局最优解，粒子群中的所有粒子根据自己找到的当前个体极值和整个粒子群共享的当前全局最优解来调整自己的速度和位置。

# 2 更新规则

公式(1)的第一部分称为记忆项，表示上次速度大小和方向的影响；公式(1)的第二部分称为自身认知项，是从当前点指向粒子自身最好点的一个矢量，表示粒子的动作来源于自己经验的部分；公式(1)的第三部分称为群体认知项，是一个从当前点指向种群最好点的矢量，反映了粒子间的协同合作和知识共享。粒子就是通过自己的经验和同伴中最好的经验来决定下一步的运动。以上面两个公式为基础，形成了PSO的标准形式。

公式(3)中的称为惯性因子，其值越大，全局寻优能力越强，局部寻优能力越弱。

# 3 算法流程图



# 4 具体代码说明

def \_\_init\_\_(self, func, n\_dim=None, pop=40, max\_iter=150, lb=-1e5, ub=1e5, w=0.8, c1=0.5, c2=0.5,constraint\_eq=tuple(), constraint\_ueq=tuple(), verbose=False,dim=None):

n\_dim = n\_dim or dim # support the earlier version

self.func = func\_transformer(func) #想要优化的函数

self.n\_dim = n\_dim #func中的变量数

self.pop = pop # 粒子数

self.max\_iter = max\_iter #iter迭代的最大值

self.lb, self.ub = np.array(lb) \* np.ones(self.n\_dim), np.array(ub) \* np.ones(self.n\_dim)

#lb：func中每个变量的下界，ub：func中每个变量的上界

self.w = w # 惯性因子

self.cp, self.cg = c1, c2 # 分别控制局部最佳、全局最佳的参数

self.constraint\_ueq = constraint\_ueq #不平等的约束

self.verbose = verbose #是否打印每个iter的结果

assert self.n\_dim == len(self.lb) == len(self.ub), 'dim == len(lb) == len(ub) is not True'

assert np.all(self.ub > self.lb), 'upper-bound must be greater than lower-bound'

self.has\_constraint = bool(constraint\_ueq)

self.is\_feasible = np.array([True] \* pop)

self.X = np.random.uniform(low=self.lb, high=self.ub, size=(self.pop, self.n\_dim))

v\_high = self.ub - self.lb

self.V = np.random.uniform(low=-v\_high, high=v\_high, size=(self.pop, self.n\_dim))

#粒子的速度

self.Y = self.cal\_y() #y = f(x) 对于所有粒子

self.pbest\_x = self.X.copy() #历史上每个粒子的最佳位置

self.pbest\_y = np.array([[np.inf]] \* pop) #历史上最好的粒子图像

self.gbest\_x = self.pbest\_x.mean(axis=0).reshape(1, -1) #所有粒子的全局最佳位置

self.gbest\_y = np.inf #所有粒子的全局最佳y

self.gbest\_y\_hist = [] # 每次迭代的gbest\_y

self.update\_gbest()

# 记录详细的值

self.record\_mode = False

self.record\_value = {'X': [], 'V': [], 'Y': []}

self.best\_x, self.best\_y = self.gbest\_x, self.gbest\_y # 历史原因，将被弃用

def check\_constraint(self, x): # 收集所有不相等的约束函数

for constraint\_func in self.constraint\_ueq:

if constraint\_func(x) > 0:

return False

return True

def update\_V(self): # 更新速度

r1 = np.random.rand(self.pop, self.n\_dim)

r2 = np.random.rand(self.pop, self.n\_dim)

self.V = self.w \* self.V + \

self.cp \* r1 \* (self.pbest\_x - self.X) + \

self.cg \* r2 \* (self.gbest\_x - self.X)

def update\_X(self): # 更新位置

self.X = self.X + self.V

self.X = np.clip(self.X, self.lb, self.ub)

def cal\_y(self): #对X中的每一个x计算y

self.Y = self.func(self.X).reshape(-1, 1)

return self.Y

def update\_pbest(self): # 更新局部最优

self.need\_update = self.pbest\_y > self.Y

for idx, x in enumerate(self.X):

if self.need\_update[idx]:

self.need\_update[idx] = self.check\_constraint(x)

self.pbest\_x = np.where(self.need\_update, self.X, self.pbest\_x)

self.pbest\_y = np.where(self.need\_update, self.Y, self.pbest\_y)

def update\_gbest(self): # 更新全局最优

idx\_min = self.pbest\_y.argmin()

if self.gbest\_y > self.pbest\_y[idx\_min]:

self.gbest\_x = self.X[idx\_min, :].copy()

self.gbest\_y = self.pbest\_y[idx\_min]

def recorder(self):

if not self.record\_mode:

return

self.record\_value['X'].append(self.X)

self.record\_value['V'].append(self.V)

self.record\_value['Y'].append(self.Y)

def run(self, max\_iter=None, precision=1e-7, N=20):

#如果precision为None，它将运行max\_iter的步数

#如果精度是一个浮点数，如果连续的N与pbest的差小于precision，循环将停止

self.max\_iter = max\_iter or self.max\_iter

c = 0

for iter\_num in range(self.max\_iter):

self.update\_V()

self.recorder()

self.update\_X()

self.cal\_y()

self.update\_pbest()

self.update\_gbest()

if precision is not None:

tor\_iter = np.amax(self.pbest\_y) - np.amin(self.pbest\_y)

if tor\_iter < precision:

c = c + 1

if c > N:

break

else:

c = 0

if self.verbose:

print('Iter: {}, Best fit: {} at {}'.format(iter\_num, self.gbest\_y, self.gbest\_x))

self.gbest\_y\_hist.append(self.gbest\_y)

self.best\_x, self.best\_y = self.gbest\_x, self.gbest\_y

return self.best\_x, self.best\_y

fit = run

# 5 总结

粒子群算法的特质：

①是一类不确定算法。不确定性体现了自然界生物的生物机制，并且在求解某些特定问题方面优于确定性算法。

②是一类概率型的全局优化算法。非确定算法的优点在于算法能有更多机会求解全局最优解。

③不依赖于优化问题本身的严格数学性质。

④是一种基于多个智能体的仿生优化算法。粒子群算法中的各个智能体之间通过相互协作来更好的适应环境，表现出与环境交互的能力。

⑤具有本质并行性。包括内在并行性和内含并行性。

⑥具有突出性。粒子群算法总目标的完成是在多个智能体个体行为的运动过程中突现出来的。

⑦具有自组织和进化性以及记忆功能，所有粒子都保存优解的相关知识。

⑧具有稳健性。稳健性是指在不同条件和环境下算法的实用性和有效性。

⑨粒子群算法的缺点：数学理论基础还不够牢固，算法的收敛性还需要讨论。