**计算智能 作业4**

**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：

多尺度协同变异的粒子群优化算法的流程如下:

(1) 种群的初始化:随机初始化微粒的速度、位置、全局优解,计算微粒的适应度,同时作为微粒的个体最优位置.反复进行 N 次,共生成 N 个初始微粒群;

(2) 对每个微粒,比较其适应度和它经历过的好位置适应度,如果更好,则更新该微粒好位置;

(3) 对每个微粒,比较其适应度和群体所经历的好位置的适应度,如果更好,则更新全局好位置;

(4) 根据公式

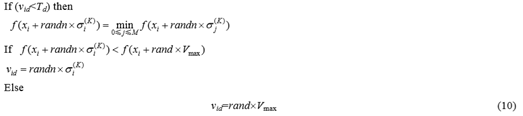
PSO 进化公式调整微粒的速度;

(5) 利用公式

在这里插入图片描述

判断微粒是否需要逃逸;

(6) 若满足逃逸条件,则利用公式

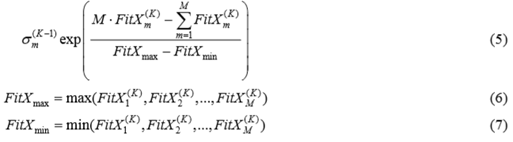


进行逃逸;

(7) 更新微粒的位置;

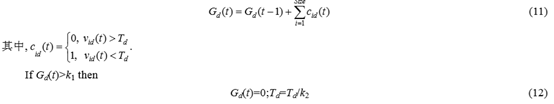
(8) 根据公式

在这里插入图片描述



计算多尺度变异算子;

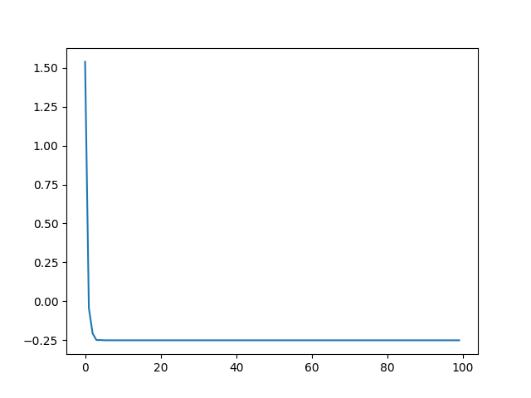
(9) 根据公式



计算每一维速度的阈值;

(10) 判断终止条件,若满足则终止,否则转到步骤(2)

1.测试Tablet函数，函数形式如下,其最小值为0，在X=[0,0,0,0,0…]时取到。 2.测试Quadric函数

图表

中度可信度描述已自动生成

图表 1Tablet Function 图表 2 Quadratic function

3.测试Rosenbrock函数

形状

描述已自动生成

图表 3 Rosenbrock function

图形用户界面

描述已自动生成

import pandas as pd

import numpy as np

import matplotlib.pylab as plt

import math

import sys

algo = 0

rang = 0

n = 50 # 个体的数量

m = 2 # 问题的维度

swarm = [] # list of swarm

swarm\_bset\_id = 0 # 最好个体的id

swarm\_best\_pos = [] # 最好个体的位置

swarm\_best\_fitness = 0 # 最好个体的适应值

c1 = 1.2 # 学习因子1

c2 = 1.4 # 学习因子2

sigma = [] # M个多尺度高斯变异算子方差

standard\_deviation = [] # M个多尺度高斯变异算子标准差

M = 10

P = int(n / M) # 每个子群个体个数

K = 0 # 迭代次数

FitXm = [] # 每个子群的适应度值

Td = [] # 逃逸点阈值

Vmax = rang # 全局最大速度

G = [] # 逃逸次数记录

k1 = 5

k2 = 10

def algo\_choose(name):

global algo, rang

if name == 1:

print('Tablet Function ')

algo = 1

rang = 500

if name == 2:

print('Quadratic function')

m = 1 # 二次函数只有一个变量-0.25[1.5]

algo = 2

rang = 100

if name == 3:

print('Rosenbrock function')

algo = 3 # 完美结果0 [1,1,1,1,1,1]

rang = 5

if name == 4:

print('Griewank Function')

algo = 4 # 计算精度不够，0 [0,0,0,0,0]

rang = 600

if name == 5:

print('Rastrigin Funtion')

algo = 5 # 0[0,0,0,0,0]

rang = 5.12

if name == 6:

print('Schaffer\'s F7 Problem')

algo = 6 # 0[0,0,0,0]

rang = 100

if name == 7:

print('Generalized Schwefel’s Problem best pos')

algo = 7

rang = 500

def cal\_fitness(x):

if algo == 1:

sum = 0

for i in range(1, m):

sum = sum + x[i] \* x[i]

return sum + x[0] \* x[0] \* 1000

if algo == 2:

sum = 0

a = 1

b = -3

c = 2

return a \* x[0] \* x[0] + b \* x[0] + c

if algo == 3:

sum = 0

for i in range(m - 1):

sum = sum + (100 \* (x[i + 1] - x[i] \* x[i]) \* (x[i + 1] - x[i] \* x[i]) + (x[i] - 1) \* (x[i] - 1))

return sum

if algo == 4:

sum1 = 0

sum2 = 1

for i in range(m):

sum1 = sum1 + x[i] \* x[i] / 4000

sum2 = sum2 \* math.cos(x[i] / math.sqrt(i + 1))

return sum1 - sum2 + 1

if algo == 5:

sum = 0

for i in range(m):

sum = sum + x[i] \* x[i] - 10 \* math.cos(2 \* math.pi \* x[i]) + 10

return sum

if algo == 6:

sum = 0

normalizer = 1.0 / float(len(x) - 1)

for i in range(len(x) - 1):

si = math.sqrt(x[i] \*\* 2 + x[i + 1] \*\* 2)

sum += (normalizer \* math.sqrt(si) \* (math.sin(50 \* si \*\* 0.20) + 1)) \*\* 2

return sum

if algo == 7:

sum = 0

for i in range(m):

sum = sum - x[i] \* math.sin(math.sqrt(abs(x[i])))

return sum

class Swarm:

def \_\_init\_\_(self):

self.pos = []

self.speed = []

self.fitness = 0

self.best\_fitness = 0

self.best\_pos = []

for i in range(m):

self.pos.append(np.random.uniform(-rang, rang))

for i in range(m):

# self.speed.append(np.random.uniform(-1,1)\*rang/100)#随机初始速度

self.speed.append(0) # 初始速度为0

self.fitness = cal\_fitness(self.pos)

self.best\_fitness = self.fitness

self.best\_pos = list.copy(self.pos)

def info(self):

print('pos', self.pos)

print('speed', self.speed)

print('fitness', self.fitness)

print('best\_pos', self.best\_pos)

def refresh\_memory(self): # 每次个体更新适应度及全局最优适应度。

global swarm\_best\_fitness, swarm\_bset\_id, swarm\_best\_pos

self.fitness = cal\_fitness(self.pos)

if (self.fitness < self.best\_fitness): # 比个体极值好，则更新记忆

self.best\_fitness = self.fitness

self.best\_pos = list.copy(self.pos)

if (self.fitness < swarm\_best\_fitness):

swarm\_best\_fitness = self.fitness

swarm\_best\_pos = list.copy(self.pos)

def refresh\_pos(self): # 跟新个体自己的速度和位置,以及适应值

for i in range(m):

self.speed[i] = self.speed[i] + c1 \* np.random.uniform() \* (

self.best\_pos[i] - self.pos[i]) + c2 \* np.random.uniform() \* (swarm\_best\_pos[i] - self.pos[i])

next\_pos = self.pos[i] + self.speed[i]

if (-rang < next\_pos and next\_pos < rang):

self.pos[i] = next\_pos

else:

# print('out of range')

pass

def escape(self, group): # 判断是否可以逃逸################biaozhuncha

temp1 = list.copy(self.pos)

temp2 = list.copy(self.pos)

for i in range(m):

temp1[i] = temp1[i] + np.random.uniform() \* standard\_deviation[group]

temp2[i] = temp2[i] + np.random.uniform() \* Vmax

temp1 = cal\_fitness(temp1)

temp2 = cal\_fitness(temp2)

for i in range(m):

if (self.speed[i] < Td[i]):

G[i] = G[i] + 1 # 记录全局逃逸次数

if (temp1 < temp2):

self.speed[i] = standard\_deviation[i] \* np.random.uniform()

else:

self.speed[i] = np.random.uniform() \* Vmax

G[i] = G[i] + 1 # 记录全局逃逸次数

def update\_swarm\_best(): # 找到全局最好的粒子,更新全局最优解和全局极值，写成函数方便切换最大最小

global swarm\_best\_fitness, swarm\_bset\_id, swarm\_best\_pos

for i in range(n):

if (swarm[i].best\_fitness < swarm\_best\_fitness):

# print('update at ',i,'best\_fitness==',swarm\_best\_fitness)

swarm\_best\_fitness = swarm[i].best\_fitness

swarm\_best\_pos = list.copy(swarm[i].best\_pos)

swarm\_bset\_id = i

return

def init():

global swarm\_best\_fitness

[swarm.append(Swarm()) for i in range(n)] # 初始化粒子群

swarm\_best\_fitness = swarm[0].best\_fitness

# update\_swarm\_best()

swarm\_best\_pos = list.copy(swarm[0].best\_pos)

for i in range(M):

sigma.append(2 \* rang)

for i in range(m):

Td.append(0.5)

G.append(0)

for i in range(M):

FitXm.append(0)

standard\_deviation.append(0)

def sort\_swarm(): # 无法使用排序算法，手写个冒泡

for i in range(n):

for j in range(0, n - i - 1):

if (swarm[j].fitness < swarm[j + 1].fitness):

swarm[j], swarm[j + 1] = swarm[j + 1], swarm[j]

def cal\_FitXm(): # 计算每个子群适应度

sort\_swarm()

for i in range(M):

sum = 0

for j in range(i \* P, i \* P + P):

sum = sum + swarm[j].fitness

FitXm[i] = sum

def cal\_standard\_deviation(): # 计算

FitX\_max = max(FitXm)

FitX\_min = min(FitXm)

for i in range(M):

standard\_deviation[i] = sigma[i] \* math.exp(((M \* FitXm[i]) - (sum(FitXm))) / (FitX\_max - FitX\_min))

while (standard\_deviation[i] > rang / 2): # 对标准差加以限制

standard\_deviation[i] = abs(rang / 2 - standard\_deviation[i])

def update\_Td():

for i in range(m):

if (G[i] > k1):

G[i] = 0

Td[i] = Td[i] / 2

def main(a):

K = 0 # 循环次数控制

algo\_choose(a)

init()

fit\_list = []

temp = 0

while (True):

for i in range(n):

swarm[i].refresh\_memory()

for i in range(n):

swarm[i].refresh\_pos()

cal\_FitXm()

cal\_standard\_deviation()

for i in range(n):

group = int(i / (n / M))

# print(group)

swarm[i].escape(group)

update\_Td()

print(swarm\_best\_fitness, swarm\_best\_pos)

# if (#精度控制条件#):

# break #到达一定精度退出程序

fit\_list.append(swarm\_best\_fitness)

# plt.plot(fit\_list)

# plt.draw()

# plt.pause(0.01)

# plt.clf()

K = K + 1

temp = swarm\_best\_fitness

if K == 100: # 到达一定次数退出程序

plt.plot(fit\_list)

plt.draw()

plt.pause(0.01)

plt.clf()

break;

main(3)