Guide of Python

内容概要： 数学建模算法

创建时间： 2022/4/7 13:41　　　　**更新时间：** 2022/4/17 15:27

作者： **TwinkelStar**

PSO 粒子群算法
PSO Algorithm

1、操作系统相关环境

1）硬件环境：

- 电脑

2）软件环境：

- Python3.7(向下兼容 Python3)(程序设计语言)
- Numpy1.19.5(兼容大部分版本)(科学计算库)

3）操作系统(2 选 1)：

- Windows7
- Windows10
- Windows11

2、粒子群算法

粒子群算法（Particle Swarm Optimization, PSO）属于进化算法的一种，粒子群算法从随机解出发，先生成一组随机数，然后通过公式不断的更新随机数，迭代更新，直到找到最符合要求的值，就是我们要寻找最优解，通过适应度（也就是函数的极值）来评价解的品质（就是解的极值是否符合要求）。

1）基本原理

PSO 可以用于解决最优化问题，在 PSO 中，每个优化问题的潜在解都是搜索空间中的一个粒子，每个粒子都由一个被优化的函数决定适值，每个粒子还有一个速度决定他们的“飞行”的方向和距离。然后粒子们就会追随当前的最优粒子在解空间中搜索，粒子的更新方式如图 1 所示：

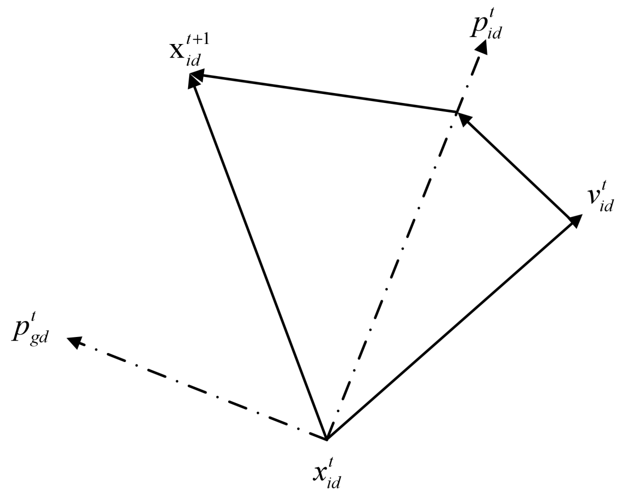

图 1.PSO 算法粒子更新
fig1.PSO Algorithm Particle Update

图 1 的内容可能比较难懂，我们可以利用抽象的思维进行想象，假设我们有一堆在空气中可移动的粒子，它们具有自己的速度以及惯性，同时还会受到空气阻力的影响，我们需要这些粒子在一个平面空间内找到一组可行解，先不讨论解的好坏，我们赋予粒子速度，让它在空间内大量的来回移动，类似于暴力枚举，寻找我们的可行解，图 2 为模拟的粒子可视化图。

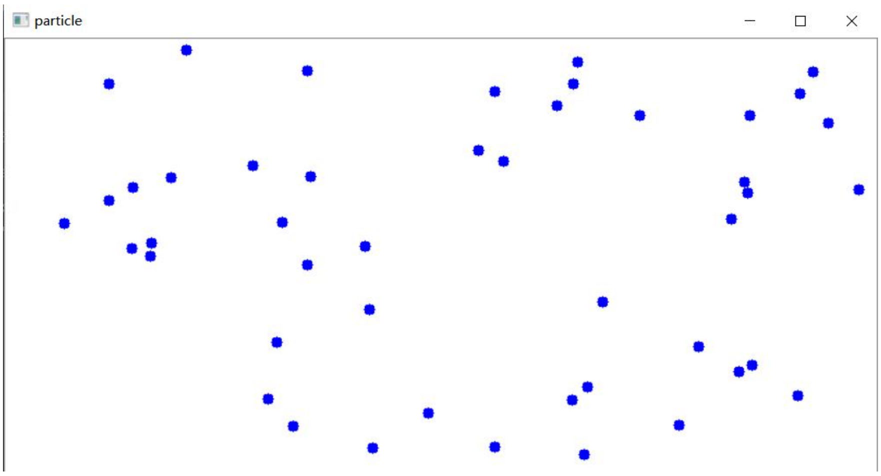

图 2.模拟算法粒子
fig2.Simulation Algorithm Particle

在图 1 中，x 表示粒子的起始位置，v 表示粒子的“飞行速度”，p 表示搜索到的粒子的最优位置。

PSO 初始化为一群随机粒子（可理解为随机解），然后通过不断的迭代更新寻找最优解，在每一次迭代中，粒子通过跟踪两个极值来对自我完成更新：一个是粒子本身找到的最优解，称为个体极值；一个是整个种群找到的最优解，称为全局极值。

2）算法思路

假设在一个 D 维的目标搜索空间中，有 N 个粒子组成一个群落其中，第 i 个粒子表示一个 D 维的向量，如（1）所示：

$$X_i = (\mathrm{x}_{i1}, \mathrm{x}_{i2}, \cdots, \mathrm{x}_{iD}), \mathrm{i} = 1, 2, \cdots, N \tag{1}$$

第 i 个粒子的“飞行”速度也是一个 D 维向量，如（2）所示：

$$V_i = (v_{i1}, v_{i2}, \cdots, v_{iD}), \mathrm{i} = 1, 2, \cdots, N \tag{2}$$

第 i 个粒子搜索到的个体极值，如（3）所示：

$$P_{best} = (p_{i1}, p_{i2}, \cdots, p_{iD}), \mathrm{i} = 1, 2, \cdots, N \tag{3}$$

整个粒子种群搜索到的最优位置的全局极值，如（4）所示：

$$g_{best} = (p_{g1}, p_{g2}, \cdots, p_{gD}) \tag{4}$$

找到两个最优值时，用公式（5）、公式（6）来对其进行更新自己的位置和速度：

$$v_{id}=w*v_{id}+c_1r_1(\mathrm{p}_{i\,\mathrm{d}}-x_{id})+c_2r_2(\mathrm{p}_{\mathrm{gd}}-x_{id}) \qquad (5)$$

$$x_{id}=x_{id}+v_{id} \qquad (6)$$

其中，C_1，C_2为学习因子，也称为加速常数，r_1，r_2为[0,1]之间的均匀随机小数。

算法的理解很简单，我们创建好我们的种群之后，将每个个体的值（一组随机值）代入我们的目标函数中，让计算机进行重复计算从而得到目标函数的值，每个个体拥有的目标函数值可以理解为个体极值，如果我们对种群的个体极值进行排序，选择最大或最小的那个值，这个值将会成为全局极值，是由整个种群的极值排序得到的。

由于粒子群算法具有高效的搜索能力，有利于得到多目标意义下的最优解，通过迭代整个解集种群，按并行的方式同时搜索多个优解。同时粒子群算法通用性较好，适合处理多种类型的目标函数和约束，并且容易与传统的优化方法相结合，改善自身的局限性，更高效的解决问题，适用于多目标优化问题。

3、例题与程序设计

1）例题

求解下列函数的最小值：

$$f(\mathrm{x})=\sum_{i=1}^{30}x_i^2+x_i-6 \qquad (7)$$

2）程序设计

基础粒子群算法的流程图如图 2 所示：

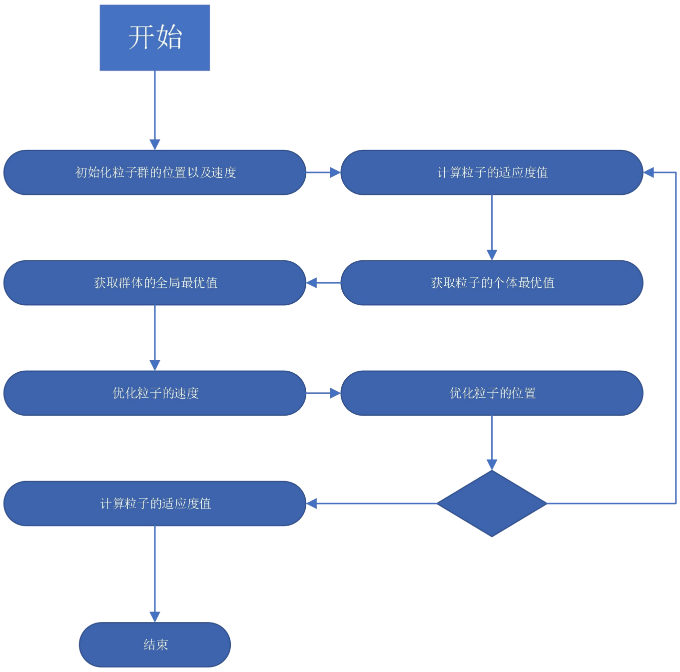

图 3.PSO 算法粒子程序框图
fig3.PSO Algorithm Particle flow chart

2）算法步骤：

为了方便程序撰写，我们对符号加以描述，如表 1 所示：

表 1.符号描述图表
table1.Symbol Description

Symbol	Description
N	群体个体数目
$P_{best}(i)$	v 的标记
$w=[w(v_i,v_j)]_{n\times m}$	待输入的加权图的带权邻接矩阵

① 初始化粒子群，N，x，y，创建三个矩阵抽象化理解为种群;

② 计算每个粒子的适应度 $F_{it}[i]$，也就是目标函数的值;

③ 对于每个粒子，用它的适应度值 $F_{it}[i]$和个体极值 $P_{best}(i)$比较，如果 $F_{it}[i]<P_{best}(i)$,则 $P_{best}(i)=F_{it}[i]$，类似于选择排序，从种群中选择一个最大或最小的值作为全局极值；

④ 对于每个粒子，用它的适应度值 $F_{it}[i]$和个体极值 $P_{best}(i)$比较，如果 $F_{it}[i]<G_{best}(i)$,则 $G_{best}(i)=F_{it}[i]$，也就是用更新之前的值和更新之后

的值做比较，符合条件则更新，否则就继续迭代；

⑤ 更新粒子的速度和位置；

⑥ 输出结果。

⑦ 输出结果。

3）Python 代码如图 4 所示：

```python
import numpy as np
import matplotlib.pyplot as plt
import cv2 as cv
def fitness(x):
    F = 0
    for i in range(x.shape[0]):
        F = F + x[i]**2 + x[i] - 6

    return F
def PSO(N,c1,c2,w,M,D):
    img = np.zeros((600,600,3))

    x = np.zeros((N,D))
    y = np.zeros((N,D))
    v = np.zeros((N,D))
    p = {}
    p_best = {}
    for i in range(N):
        for j in range(D):
            x[i,j] = np.random.randn(1)[0] #初始化位置
            v[i,j] = np.random.randn(1)[0] #初始化速度
    # 计算粒子的适应度 初始化pi 和 pg
    for i in range(N):
        p[i] = fitness(x[i,:])
        y[i,:] = x[i,:]

    pg = x[N-1,:]

    fv_lidt = []

    for i in range(N-1):
        if fitness(x[i,:]) < fitness(pg):
            pg = x[i, :]

    for t in range(M):
        for i in range(N):
            v[i,:] = w*v[i,:] + c1*np.random.rand(1)[0]*(y[i,:] - x[i,:])+c2*np.random.rand(1)*(pg-x[i,:])
            x[i,:] = x[i,:] + v[i,:]

            fv_lidt.append(fitness(x[i,:]))
            # print("fitness(pg)1: ", fitness(x[i,:]))
            # print("y[i,:]: ", y[i,:])
            if fitness(x[i,:]) < p[i]:
                p[i] = fitness(x[i,:])
                y[i, :] = x[i, :]
            if p[i] < fitness(pg):
                pg = y[i,:]

        # fv_lidt.append(fitness(pg))
        p_best[t]=fitness(pg)

        # print("p_best: ", p_best[0])
    xm = pg
    fv = fitness(pg)

    print("xm: ", pg)
    print("fv: ", fitness(pg))

    return xm, fv_lidt
```

图 4.Python PSP 算法代码

fig4.Python PSPAlgorithm Code

4）函数调用方式：

直接运行代码，运行结果如图 5 所示：

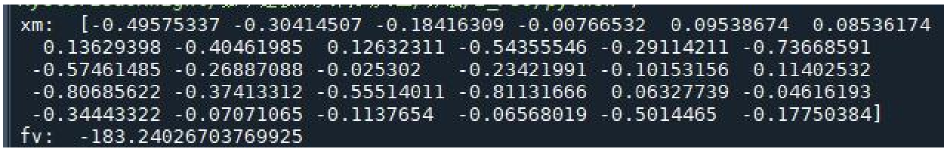

图 5.运行结果
fig5.code result

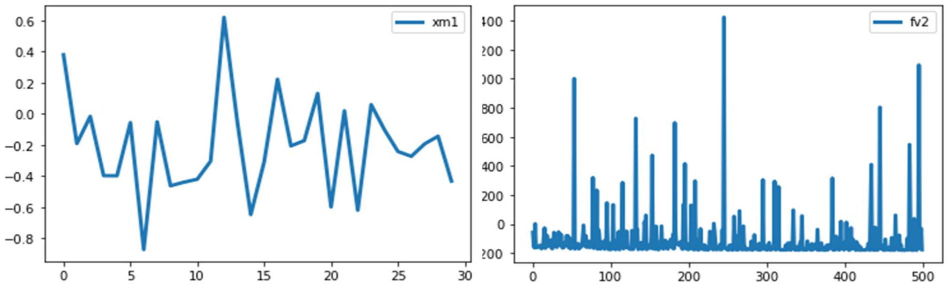

图 6.运行结果
fig6.code result

经过计算可知，目标函数的最小值为-187.5000。其中，fitness 代表目标函数，N 初始化的种群个数，w 为惯性权重，M 为迭代次数，D 为搜索空间的维度，xm1 为 x 的取值范围，fv2 为目标函数的迭代结果。需要注意的是，不一定是种群的数量和迭代的次数越高越好，学习因子的值也可以进行适当调整，当种群的数量过多时可能会出现函数不收敛的情况，因为其在搜索空间的范围扩大了，找不到全局最优值，还会提升算法排序寻找最优值（可行解）的耗费时间。