目录

湖南工商大学 算法导论 课程教案

授课题目（教学章、节或主题） **课时安排: 2学时**

第十四讲：粒子群优化算法 **授课时间 :第十四周周一第1、2节**

教学内容（包括基本内容、重点、难点）：

基本内容： （1）基本概念：粒子群算法、研究历史；

（2）粒子群算法基本思想、参数；

（3）粒子群算法步骤与流程图；

（4）粒子群算法求解非凸问题.

教学重点、难点：重点为粒子群算法的原理、粒子群算法的程序设计

教学媒体的选择：本讲使用大数据分析软件Jupyter教学，Jupyter集课件、Python程序运行、HTML网页制作、Pdf文档生成、Latex文档编译于一身，是算法导论课程教学的最佳选择。

板书设计：黑板分为上下两块，第一块基本定义，推导证明以及例子放在第二块。第一块 整个课堂不擦洗，以便学生随时看到算法流程图以及基本算法理论等内容。

课程过程设计： （1）讲解基本算法理论； （2）举例说明； （3）程序设计与编译； （4）对本课堂进行总 结、讨论； （5）布置作业与实验报告

第十四讲 粒子群优化算法

一. 基本概念

粒子群优化算法（Partical Swarm Optimization PSO）：

粒子群中的每一个粒子都代表一个问题的可能解，通过粒子个体的简单行为，群体内的信息交互实现问题求解的智能性。

1995年，由Kennedy和Eberhart提出。Shi和Eberhart对PSO算法的速度项引入了惯性权重w，调整PSO的参数来平衡算法的全局探测和局部开采能力。2009年张玮等在对标准粒子群算法位置期望及方差进行稳定性分析的基础上，得出了一组较好的加速因子c取值。

二. 粒子群算法基本思想

1. 优缺点

- 适用于非凸多峰函数和多目标函数的优化问题。
- PSO操作简单、易实现、收敛速度快、搜索能力强
- 易陷入局部最优

以鸟群举例：

一群鸟在某区域搜索食物。该区域只有一片食物，所有的鸟都不知道具体的位置，但它们知道自己和其他鸟相比谁离食物更近。那么每只鸟要以最快的速度找到这块食物的话，就要寻找比自己离食物更近的鸟，并且依据他的位置更新自己的位置。然后根据自己的飞行经验调整路线。最终，群里的每一只鸟都找到了这片食物。

- 每个寻优的解都被看作一只鸟，称为**粒子**。所有粒子都在一个D维空间搜索；
- 所有粒子都由一个**适应度函数**来判定目前**位置**的好坏；
- 每个粒子赋予一定的记忆功能，能记住所搜寻到的**最佳位置**；

- 每个粒子还有一个**速度**以决定**飞行距离和方向**。此速度会根据自身飞行经验和同伴飞行经验**动态调整**。

2. 粒子群算法描述

在D维空间，粒子的位置为：

$$x = \{x_1, x_2, \dots, x_D\}$$

粒子的速度为：

$$v = \{v_1, v_2, \dots, v_D\}$$

粒子i在搜索过程中，记住其最佳位置P_i，群体的最佳位置P_g。每个粒子根据自身经验P_i和群体经验P_g动态调整位置和速度:

$$x_{id} = x_{id} + v_{id}, (i = 1, 2, \dots, n; d = 1, 2, \dots, D)$$

$$v_{id} = w \cdot v_{id} + c_1 r_1 (p_i - x_{id}) + c_2 r_2 (p_g - x_{id})$$

3. 算法参数

- **种群大小：** 当n很小示会导致局部优化；当种群规模达到一定水平时，再增长无显著作用。
- **惯性因子**w**：** 用于调节粒子搜索范围。较大的可加强全局搜索能力。较小的 能加强局部搜索。
- **学习因子**c_1**和**c_2：代表粒子偏好的权值，介于$[0, 4]$, 他们的和为4.1时最佳。又称**加速因子**。
- **随机数**r_1**和**r_2：属于区间$(0, 1)$的随机数。

$$x_{id} = x_{id} + v_{id},\ (i = 1, 2, \dots, n; d = 1, 2, \dots, D)$$
$$v_{id} = w \cdot v_{id} + c_1 r_1 (p_i - x_{id}) + c_2 r_2 (p_g - x_{id})$$

- 注：经常将w从最大0.9到最小0.4动态调整

三. 标准流程

标准PSO算法的流程：

Step1: 初始化一群微粒(群体规模为m)，包括随机位置和 速度；

Step2: 评价每个微粒的适应度；

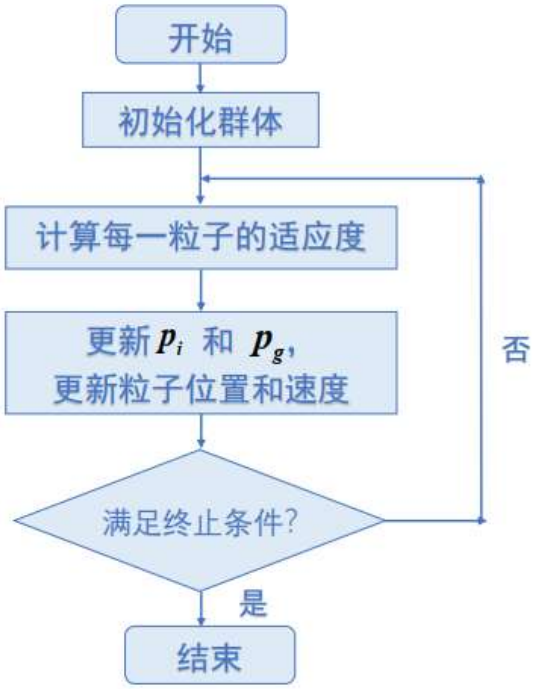
Step3: 对每个微粒，将其适应值与其经过的最好位置 pbest作比较，如果较好，则将其作为当前的 最好位置pbest;

Step4: 对每个微粒，将其适应值与其经过的最好位置 gbest作比较，如果较好，则将其作为当前的 最好位置gbest;

Step5: 根据(2)、(3)式调整微粒速度和位置；

Step6：未达到结束条件则转Step2。

迭代终止条件根据具体问题一般选为最大迭代次数Gk或(和)微粒群迄今为止搜索到的最优位置满足预定最小适应阈值。



四. 算法举例

例 1：求解如下四维Rosenbrock函数的优化问题

$$\min\ f(x) = \sum_{d=1}^{3}[100(x_{d+1} - x_d^2)^2 + (x_d - 1)^2]$$

$$x_d \in [-30, 30], \quad d = 1, 2, 3, 4$$

解 种群大小：即算法中粒子的数量 n=5

编码：问题的维数为4，所以每个粒子的位置和速度均为4维的实数向量。

设定粒子的最大速度为： $V_{max} = 60$

$$x = \{x_1, x_2, \dots, x_D\}, v = \{v_1, v_2, \dots, v_D\}$$

对粒子群进行随机初始化（本例中D=4）包括随机初始化各粒子的位置和速度

初始位置:

$$x_1^{(0)} = \{21.721, -9.13677, 6.62244, 3.84079\}$$
$$x_2^{(0)} = \{-13.5001, -23.6131, 17.4462, -29.0515\}$$
$$x_3^{(0)} = \{-29.6563, -0.871811, -27.8912, 17.7425\}$$
$$x_4^{(0)} = \{23.6218, -16.3885, -22.7019, 25.40333\}$$
$$x_5^{(0)} = \{-28.0992, 22.6482, 0.675616, -8.43752\}$$

初始速度:

$$v_1^{(0)} = \{-19.9048, 29.562, -22.104, -5.45346\}$$
$$v_2^{(0)} = \{-20.5922, -28.6944, -26.3216, 19.0615\}$$
$$v_3^{(0)} = \{-7.83576, -55.7173, -40.9177, 28.255\}$$
$$v_4^{(0)} = \{-11.6373, -41.0138, 17.7311, -14.87\}$$
$$v_5^{(0)} = \{17.561, -13.5365, 51.2722, -56.098\}$$

初代适应度计算:

适应度函数:

$$f(x) = \sum_{d=1}^{3}[100(x_{d+1} - x_d^2)^2 + (x_d - 1)^2]$$

$$f(x_1^{(0)}) = 2.38817 \times 10^7 \leftarrow \text{初始种群最优解}$$
$$f(x_2^{(0)}) = 4.45306 \times 10^7$$
$$f(x_3^{(0)}) = 1.35376 \times 10^8$$
$$f(x_4^{(0)}) = 6.56888 \times 10^7$$
$$f(x_5^{(0)}) = 8.50674 \times 10^7$$

更新位置和速度：

群体最优解：$P_g = x_1^{(0)}$

个体历史最优解：$P_i = x_i^{(0)}, (i = 1, 2, 3, 4, 5)$

计算第一代粒子的位置和速度，取：$w = 1, c_1 = c_2 = 2$

$$x_{id} = x_{id} + v_{id}\ , (i = 1, 2, 3, 4, 5; d = 1, 2, 3, 4)$$

$$v_{id} = v_{id} + 2r_1(p_i - x_{id}) + 2r_2(p_g - x_{id})$$

第一代粒子群速度更新：

$$v_1^{(0)} = \{-19.9048, 29.562, -22.104, -5.45346\}$$
$$v_2^{(0)} = \{40.0498, -3.76972, -44.9573, 60\}$$
$$v_3^{(0)} = \{14.8665, -59.3694, -25.667, 22.1122\}$$
$$v_4^{(0)} = \{-13.843, -32.4824, , 17.731, -39.892\}$$
$$v_5^{(0)} = \{60, -60, 60, -36.7909\}$$

第一代粒子的位置更新：

$$x_1^{(1)} = \{1.81621\ ,\ 20.4252\ ,\ -15.4816\ ,\ -1.61267\}$$
$$x_2^{(1)} = \{26.5497\ ,\ -27.3829, -27.5112, 30.9485\}$$
$$x_3^{(1)} = \{-29.6563, -0.871811, -27.8912, 17.7425\}$$
$$x_4^{(1)} = \{23.6218, -16.3885, -22.7019, 25.40333\}$$
$$x_5^{(1)} = \{-28.0992, 22.6482, 0.675616, -8.43752\}$$

第一代粒子群适应度计算：

适应度函数：

$$f(x) = \sum_{d=1}^{3}[100(x_{d+1} - x_d^2)^2 + (x_d - 1)^2]$$

$$f(x_1^{(0)}) < f(x_1^{(1)}) = 2.45726 \times 10^7 \leftarrow \text{第一代种群最优解}$$
$$f(x_2^{(0)}) < f(x_2^{(1)}) = 1.6674 \times 10^8$$
$$f(x_3^{(0)}) < f(x_3^{(1)}) = 2.16403 \times 10^9$$
$$f(x_4^{(0)}) < f(x_4^{(1)}) = 6.37125 \times 10^8$$
$$f(x_5^{(0)}) < f(x_5^{(1)}) = 1.6783 \times 10^9\$$

计算第二代粒子的位置和速度，取:

$$w_{max} = 0.9, w_{min} = 0.4, c_1 = c_2 = 2$$

$$w = w_{max} - \frac{(w_{max} - w_{min}) * t}{T_{max}}$$

重复上述步骤，将迭代进行下去。经过1000次迭代，粒子群算法得到比较好的适应值，相应的粒子位置即为近似最优解P_g，对应的适应值为目标函数近似最优值。

例 2：求解如下函数的最小值问题

$$f(x) = \sum_{i=1}^{10} x_i^2$$

解 种群大小: 即算法中粒子的数量 n=40

编码：问题的维数为10，所以每个粒子的位置和速度均为10维的实数向量。

设定最大迭代次数为： $MaxDT = 1000$

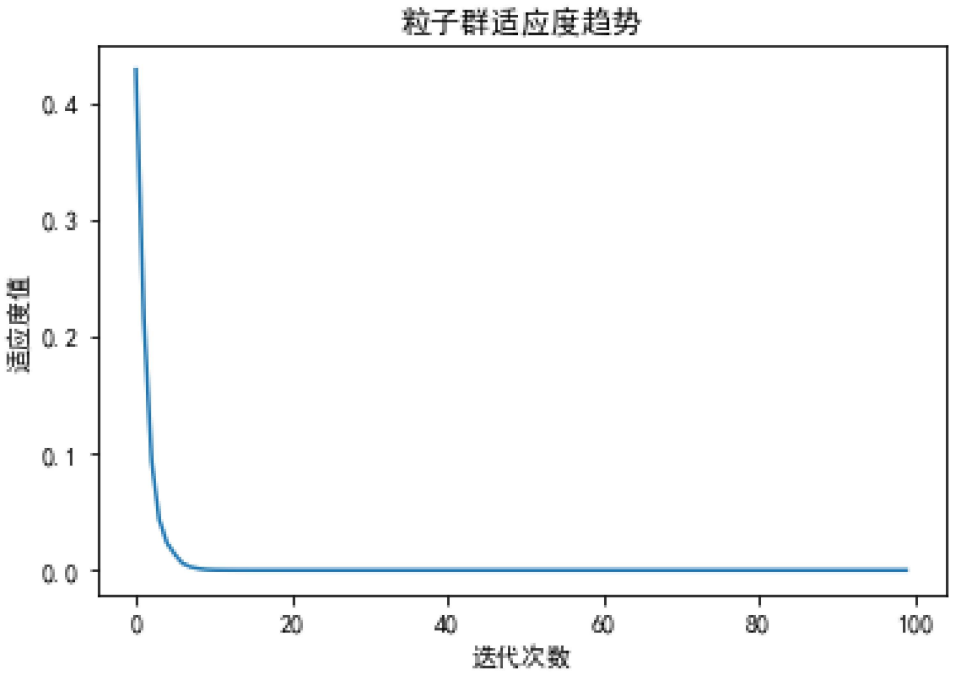
python程序代码

In [43]:

```
import random
import matplotlib.pyplot as plt

plt.rcParams['font.sans-serif'] = ['SimHei']  # 步骤一（替换sans-serif字体）
plt.rcParams['axes.unicode_minus'] = False

class PSO(object):
    def __init__(self):
        self.x_bound = [-1, 1]
        self.T = 100
        self.w = 0.15
        self.N = 1000
        self.dim = 10
        self.c1 = 1.5
        self.c2 = 1.5
        self.pso_main()

    def fun(self, x):
        result = 0
        for i in x:
            result = result + pow(i, 2)
        return result

    def pso_main(self):
        x = []
        v = []
        for j in range(self.N):
            x.append([random.random() for i in range(self.dim)])
            v.append([random.random() for m in range(self.dim)])
        fitness = [self.fun(x[j]) for j in range(self.N)]
        p = x
        best = min(fitness)
        pg = x[fitness.index(min(fitness))]
        best_all = []
        for t in range(self.T):
            for j in range(self.N):
                for m in range(self.dim):
                    v[j][m] = self.w * v[j][m] + self.c1 * random.random() * (
                            p[j][m] - x[j][m]) + self.c2 * random.random() * (
            for j in range(self.N):
                for m in range(self.dim):
                    x[j][m] = x[j][m] + v[j][m]
                    if x[j][m] > self.x_bound[1]:
                        x[j][m] = self.x_bound[1]
                    if x[j][m] < self.x_bound[0]:
                        x[j][m] = self.x_bound[0]
```

```
            fitness_ = []
            for j in range(self.N):
                fitness_.append(self.fun(x[j]))
            if min(fitness_) < best:
                pg = x[fitness_.index(min(fitness_))]
                best = min(fitness_)
            best_all.append(best)

        print('第' + str(t) + '次迭代：最优解位置在' + str(pg) + '，最优解的适
        plt.plot([t for t in range(self.T)], best_all)
        plt.ylabel('适应度值')
        plt.xlabel('迭代次数')
        plt.title('粒子群适应度趋势')
        plt.show()

if __name__ == '__main__':
    PSO()
```

第99次迭代：最优解位置在[1.978876427566799e-05, 3.658117292436997e-08, -1.0255911590284626e-06, 3.1438256713063637e-09, 2.4676083977831173e-06, 1.1216143622733329e-05, 1.56055150483144e-06, -2.8671229351567438e-06, -7.360275661549497e-06, -2.760389686424793e-06]，最优解的适应度值为：5.969884697793265e-10



Matlab程序

```
%适应度函数 fitness.m
function result=fitness(x,D)
sum=0;
for i=1:D
    sum=sum+x(i)^2;
end
result=sum;
%%%%%%%%%%%%%%%%
%主函数 PSO_1_20190812. m
clear all;
clc;
format long;
%--------给定初始化条件-----------
c1=1.4962;              %学习因子1
c2=1.4962;              %学习因子2
W_max=0.9;              %最大惯性权重
W_min=0.4;               %最小惯性权重MaxDT=1000;     %最大迭代次数
D=10;                    %搜索空间维数
N=40;                    %初始化种群个体数目
eps=10^(-6);        %设置精度
%初始化种群的个体
for i=1:N
    for j=1:D
        x(i,j)=randn;        %随机初始化位置
        v(i,j)=randn;        %随机初始化速度
    end
end
%先计算适应度，并初始化Pi和Pg
for i=1:N
    p(i)=fitness(x(i,:),D);
    y(i,:)=x(i,:);
end
pg=x(1,:);         %Pg为全局最优解
for i=2:N
    if fitness(x(i,:),D)<fitness(pg,D)
        pg=x(i,:);
    end
end
%进入主循环，按公式依次迭代
for t=1:MaxDT
    w=W_max-((W_max-W_min)*t)/MaxDT;
```

```
    for i=1:N
        v(i,:)=w*v(i,:)+c1*rand*(y(i,:)-x(i,:))+c2*rand*(pg-x(i,:));
        x(i,:)=x(i,:)+v(i,:);
        if fitness(x(i,:),D)<p(i)
            p(i)=fitness(x(i,:),D);
            y(i,:)=x(i,:);
        end
        if p(i)<fitness(pg,D);
            pg=y(i,:);
        end
    end
    pbest(t)=fitness(pg,D);
end
%------最后给出计算结果---
disp('**************************')
disp('函数的全局最有位置为：')
solution=pg
disp('最后得到的优化取值为：')
Result=fitness(pg,D)
disp('**************************')
%-----算法结果------------
```

例 3 求解如下函数最大值问题

$$max\ f(x) = 2.1(1 - x + 2x_2)exp(-\frac{x^2}{2}), \quad x \in [-5,5]$$

解 种群大小：即算法中粒子的数量 n=30

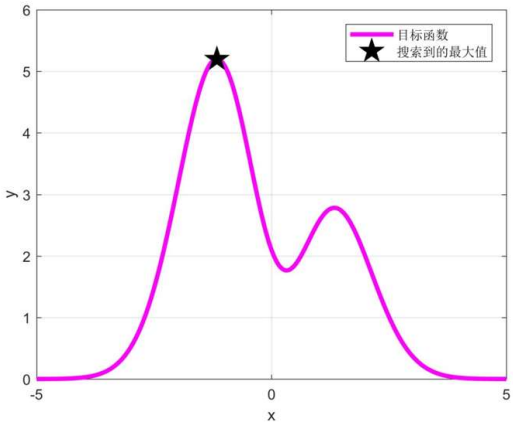
编码： 问题的维数为1，所以每个粒子的位置和速度均为1维的实数向量。

设定最大迭代次数为：$MaxNun = 100$

编写Matlab程序: PSO0812_3.m

程序运行结果（Matlab）：

```
the maximum value=5.1985
the corresponding coordinate=-1.1617
时间已过 0.455766 秒。
```



编写Python程序:

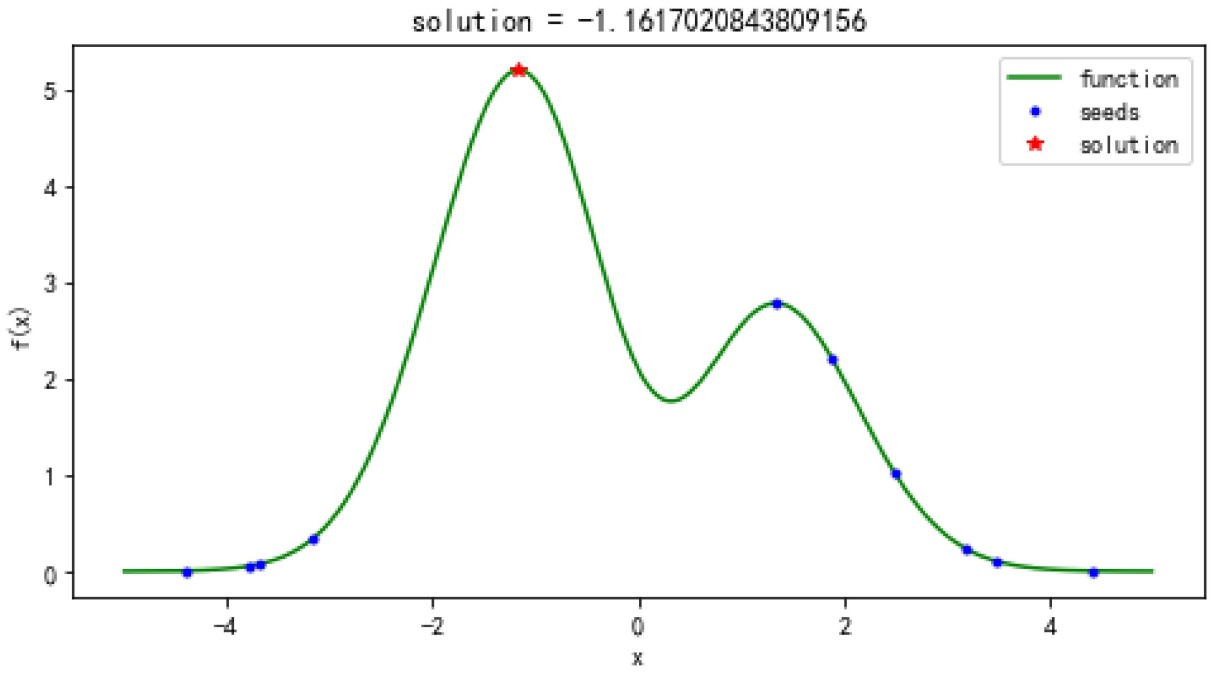
In [44]:

```
import numpy as np
import matplotlib.pyplot as plt
import random


# 定义“粒子”类
class parti(object):
    def __init__(self, v, x):
        self.v = v                      # 粒子当前速度
        self.x = x                      # 粒子当前位置
        self.pbest = x                  # 粒子历史最优位置

class PSO(object):
    def __init__(self, interval, tab='min', partisNum=10, iterMax=1000, w=1,
        self.interval = interval                                            #
        self.tab = tab.strip()                                              #
        self.iterMax = iterMax                                              #
        self.w = w                                                          #
        self.c1, self.c2 = c1, c2                                           #
        self.v_max = (interval[1] - interval[0]) * 0.1                      #
        ###########################################################################
        self.partis_list, self.gbest = self.initPartis(partisNum)
        self.x_seeds = np.array(list(parti_.x for parti_ in self.partis_list))
        self.solve()
        self.display()

    def initPartis(self, partisNum):
        partis_list = list()
        for i in range(partisNum):
            v_seed = random.uniform(-self.v_max, self.v_max)
            x_seed = random.uniform(*self.interval)
            partis_list.append(parti(v_seed, x_seed))
        temp = 'find_' + self.tab
        if hasattr(self, temp):                                             #
            gbest = getattr(self, temp)(partis_list)
        else:
            exit('>>>tab标签传参有误："min"|"max"<<<')
        return partis_list, gbest

    def solve(self):
        for i in range(self.iterMax):
            for parti_c in self.partis_list:
                f1 = self.func(parti_c.x)
                # 更新粒子速度，并限制在最大迁移速度之内
                parti_c.v = self.w * parti_c.v + self.c1 * random.random() *
                if parti_c.v > self.v_max: parti_c.v = self.v_max
```

```
                elif parti_c.v < -self.v_max: parti_c.v = -self.v_max
                # 更新粒子位置，并限制在待解空间之内
                if self.interval[0] <= parti_c.x + parti_c.v <=self.interval[
                    parti_c.x = parti_c.x + parti_c.v
                else:
                    parti_c.x = parti_c.x - parti_c.v
                f2 = self.func(parti_c.x)
                getattr(self, 'deal_'+self.tab)(f1, f2, parti_c)            #

    def func(self, x):                                                      #
        value = 2.1*(1-x+2*x**2)*np.exp(-x**2/2)
        return value

    def find_min(self, partis_list):                                        #
        parti = min(partis_list, key=lambda parti: self.func(parti.pbest))
        return parti.pbest

    def find_max(self, partis_list):
        parti = max(partis_list, key=lambda parti: self.func(parti.pbest))
        return parti.pbest

    def deal_min(self, f1, f2, parti_):
        if f2 < f1:                             # 更新粒子历史最优位置
            parti_.pbest = parti_.x
        if f2 < self.func(self.gbest):
            self.gbest = parti_.x               # 更新群体历史最优位置

    def deal_max(self, f1, f2, parti_):
        if f2 > f1:                             # 更新粒子历史最优位置
            parti_.pbest = parti_.x
        if f2 > self.func(self.gbest):
            self.gbest = parti_.x               # 更新群体历史最优位置

    def display(self):
        print('solution: {}'.format(self.gbest))
        print('value: {}'.format(self.func(self.gbest)))
        plt.figure(figsize=(8, 4))
        x = np.linspace(self.interval[0], self.interval[1], 300)
        y = self.func(x)
        plt.plot(x, y, 'g-', label='function')
        plt.plot(self.x_seeds, self.func(self.x_seeds), 'b.', label='seeds')
        plt.plot(self.gbest, self.func(self.gbest), 'r*', label='solution')
        plt.xlabel('x')
        plt.ylabel('f(x)')
        plt.title('solution = {}'.format(self.gbest))
        plt.legend()
        plt.savefig('PSO.png', dpi=500)
        plt.show()
```

```
        plt.close()


if __name__ == '__main__':
    PSO([-5, 5], 'max')
```

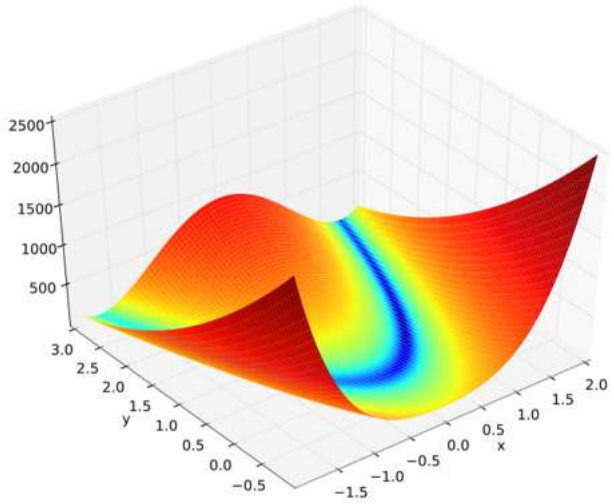
```
solution: -1.1617020843809156
value: 5.198476768427025
```



二元Rosenbrock函数

在数学最优化中，Rosenbrock函数是一个用来测试最优化算法性能的非凸函数，由H. H. Rosenbrock在1960年提出。也称为香蕉函数。

$$f(x,y) = (1-x)^2 + 100(y-x^2)^2$$



引用及参考：

[1]《Python数据结构与算法分析》布拉德利.米勒等著，人民邮电出版社，2019年9月.

[2] https://blog.csdn.net/weixin_42018258/article/details/80670067 (https://blog.csdn.net/weixin_42018258/article/details/80670067)

课后练习

1. 写出粒子群算法的完整Python代码或 C语言代码。

2. 写出粒子群算法的标准流程与实现步骤，举例说明。

3. 利用粒子群算法求解非凸问题，可视化算法结果，研究收敛速度与时间代价。

讨论、思考题、作业：

参考资料（含参考书、文献等）：算法导论. Thomas H. Cormen等，机械工业出版社，2017.

授课类型（请打√）：理论课□ 讨论课□ 实验课□ 练习课□ 其他□

教学过程设计（请打√）：复习□ 授新课□ 安排讨论□ 布置作业□

教学方式（请打√）：讲授□ 讨论□ 示教□ 指导□ 其他□

教学资源（请打√）：多媒体□ 模型□ 实物□ 挂图□ 音像□ 其他□

填表说明：1、每项页面大小可自行添减；2、教学内容与讨论、思考题、作业部分可合二为一。