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**

蛮力法

Brute Force

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、蛮力法

蛮力法是一种简单直接地解决问题的方法(暴力求解)，常常直接基于问题的描述和所涉及的概念定义。注意，这里的“力”是指计算机的计算的运算能力。一般来说，蛮力策略常常是最容易应用的办法。虽然巧妙和高效的算法很少来自于蛮力法，但我们不应该忽略它作为

一种重要的算法设计策略的地位。

第一，蛮力法可以解决广阔领域的各种问题。实际上，它可能是唯一一种几乎什么问题都能解决的一般性方法。

第二，对于一些重要的问题（如排序、查找、字符串匹配等）来说，蛮力法可以产生一些合理的算法。

第三，如果要解决的问题实例不多，而且蛮力法可以用一种能够接受的速度对实例求解，那么设计一个更高效算法所花费的代价很可能是不值得的。

第四，即使效率通常很低，仍可使用蛮力法解决一些小规模的问题实例。第五，蛮力法可以作为研究或教学目的服务，如可以以之为准绳，来衡量同样问题的更高效算法（如计算最坏时间复杂度）。

1）设计思想

蛮力法是指采用遍历（扫描）技术，即采用一定的策略将待求解问题的所有元素依次处理一次，从而找出问题的解。依次处理所有元素是蛮力法的关键，为了避免陷入重复试探，应保证处理过的元素不再被处理。

蛮力法本质是先有策略地穷举，然后验证。简单来说，就是列举问题所有可能的解，然后去看看是否满足题目要求，是一种逆向解题方式。

3、算法分析

1）算法优点

- 逻辑简单清晰，根据逻辑编写的程序简单清晰。
- 对于一些需要高正确性的重要问题，它产生的算法一般而言复杂度高但合理的。
- 解决问题的领域广。适用于一些规模较小，时间复杂度要求不

高的问题。

● 可以作为其他高校算法的衡量标准，即作为被比较的例子。

2）算法缺点

算法的主要缺点就算缺少人的思考，算法设计简单但往往缺乏效率，在所有的解空间中通过搜索解。

4、例题

1）在象棋算式里，不同的棋子代表不同的数，有以下算式，设计一个算法求这些棋子各代表哪些数字。

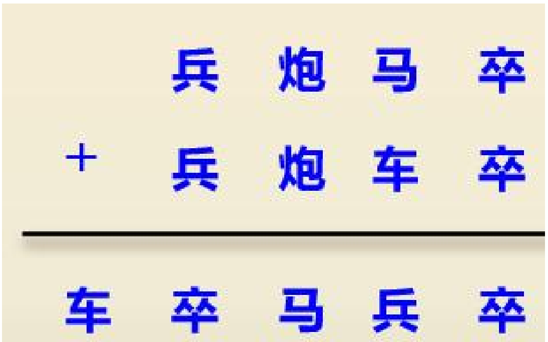

图 1.案例 1

fig1. Example1 one

直接采用穷举法的思想计算，对于五个棋子的取值分别进行枚举，然后判断取值是否成立，满足苏式要求，运行代码 example1.py 代码如图 2 所示：

```python
#分别穷举兵,炮,马,卒,车的各种可能  a,b,c,d,e
for a in range(10):
    for b in range(10):
        for c in range(10):
            for d in range(10):
                for e in range(10):
                    #每个棋子的取值不同 做判断
                    if a != b and  a != c and a != d and  a != e and \
                    b != c and b != d and b != e and c != d and c != e and d != e:
                        #将竖式计算出来 判断等式是否相等，相等则输出结果
                        m1 = a * 1000 + b * 100 + c * 10 + d
                        m2 = a * 1000 + b * 100 + e * 10 + d
                        s = e * 10000 + d * 1000 + c * 100 + a * 10 + d
                        if s == m1 + m2:
                            print("兵",a)
                            print("炮",b)
                            print("马",c)
                            print("卒",d)
                            print("车",e)
```

图 2.案例 1 代码

fig2. Example one code

最后的运行结果为：

兵：5；炮：2；马：4；卒：0；车：1。

验算之后等式成立。

2）求解函数$(x_1-5)^2+(x_2-7)^2$ 的最小值，x_1、x_2 的取值范围为[0,10]。

从题目中分析不难得知，这是函数是一条抛物线，当 x_1 取 5，x_2 取 7 时，函数取得最小值。使用蛮力法的思想，生成两组样本数据，范围在[0,10]之间，然后喂入函数最后寻找最小值对应的取值即可，运行代码 example2.py，代码如图 3 所示：

```
import numpy as np
import matplotlib.pyplot as plt
def f(x1, x2):
    #目标函数
    return np.square((x1 - 5)) + np.square((x2 - 7))
#生成一组等差数列，范围0-10，个数100个
x1 = np.linspace(0,10,100)
x2 = np.linspace(0,10,100)

l = []
xx = []
#迭代开始
for i in range(x1.shape[0]):
    for j in range(x2.shape[0]):
        y = f(x1[i],x2[j])
        xx.append(y)
        l.append([y,x1[i],x2[j]])
plt.plot(xx, lw=3, label='l')
plt.legend()
plt.show()
l = np.array(l)
print("最小值为: ", np.min(l[:,0]))
print("x1的取值为: ", l[np.argmin(l[:,0]),1])
print("x2的取值为: ", l[np.argmin(l[:,0]),2])
```

图 3.案例 2 代码

fig3. Example two code

最后的运行结果为：

最小值为： 0.0034690337720640393；

x1 的取值为： 4.94949494949495；

x2 的取值为： 6.96969696969697。

当 x1 近似取 5，x2 近似取 7 时，函数有最极值，最小值为 0

绘制函数图形，如图 4 所示：

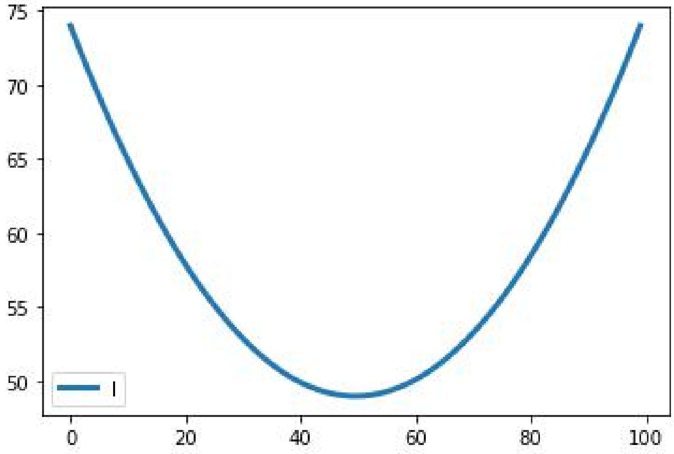

图 4.案例 2 函数图形

fig4. Example two function img

3）考虑一个线性规划的小规模案例，使得目标函数$3x+5y$取得极大值，尝试用蛮力法求解，约束条件为：

$$x+y\leq 4 \tag{1}$$

$$x+3y\leq 6 \tag{2}$$

$$x\geq 0\text{且}y\geq 0 \tag{3}$$

运行程序 example3.py，代码如图 5 所示：

```
import matplotlib.pyplot as plt
#初始化 x,y的取值
x = 0
y = 0
result = []
epoch = 1000
#目标函数
def f(x,y):
    return 3 * x + 5 * y
#开始迭代
for i in range(epoch):
    for j in range(epoch):
        x,y = i/100,j/100
        #约束条件
        if x + y <= 4 and x + 3 * y <= 6:
            result.append([f(x,y),x,y])

result = np.array(result)
plt.plot(result[:,0], lw=3, label='result')
plt.legend()
plt.show()
print("最大值为: ",np.max(result[:,0]))
print("x的值为: ",result[np.argmax(result[:,0]),1])
print("y的值为: ",result[np.argmax(result[:,0]),2])
```

图 5.案例 3 代码

fig5. Example three code

绘制图形，如图 6 所示

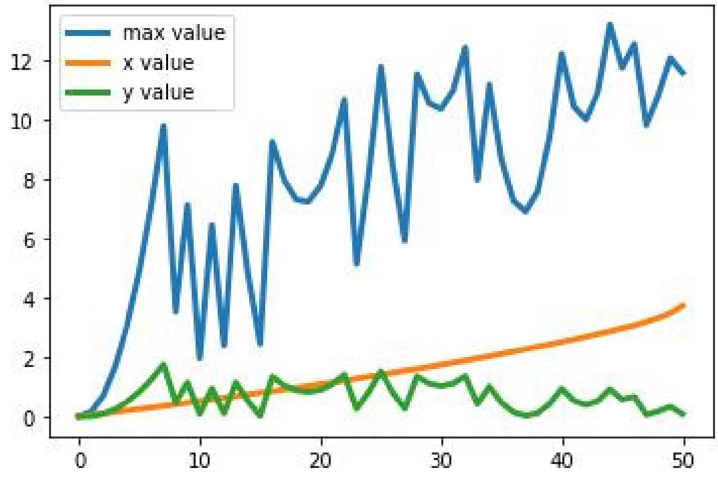

图 6.案例 3 函数图形

fig6. Example three function img

从图像以及结果得知，函数在第 45347 次迭代时产生的最大值，最大值为：14.0，x 的值为：3.0，y 的值为：1.0。

但蛮力法的这种暴力枚举的方法在时间选择上不可取，尽管理论上来说蛮力法可以穷举出所有的可能性，但对于一些规划问题和图论问题，需要在短时间内完成方案的选择，蛮力法的表现就不尽人意，并且，从图 6 中可以看出，蛮力法求解解空间并不问题，只是在解空间里面到处“试错”。

在已有的计算资源的情况下，对于一些基础的问题可以用蛮力法求解，但考虑到时间复杂度和代码的优化上来说，尽量设计出更好的程序，便于优化方案。