凸优化课程设计

09118139 王明灏

例 5.2 注水。考虑如下凸优化问题：

$$\min.\ -\sum_{i=1}^{n}\log(\alpha_i+x_i)$$
$$subject\ to\ x\succeq 0,\ 1^Tx=1,$$

1. 问题的实际应用：

该问题源自信息论，将功率分配给 n 个信道。变量x_i表示分配给第 i 个信道的发射功率，$\log(\alpha_i+x_i)$是信道的通信能力或通信速率，因此上述问题即为将值为一的总功率分配给不同的信道，使得总的信道速率最大。

2. 问题求解及实验目标分析：

对于不等式约束$x^*\succeq 0$引入$Lagrange$乘子$\lambda^*\in R^n$，对等式约束$1^Tx=1$引入一个乘子$\nu^*\in R$，我们得到如下KKT条件：

$$x^*\succeq 0,\quad 1^Tx^*=1,\quad \lambda^*\succeq 0,\quad \lambda_i^*x_i^*=0,\quad i=1,\cdots,n,$$
$$\frac{-1}{(\alpha_i+x_i^*)}-\lambda_i^*+\nu^*=0,\quad i=1,\cdots,n.$$

可以直接求解这些方程得到x^*，λ^*，ν^*。注意到λ^*在最后一个方程里是一个松弛变量①，所以可以消去。

消去之后可以得到：

$$x^*\succeq 0,\quad 1^Tx^*=1,\quad x_i^*\left(\nu^*-\frac{1}{(\alpha_i+x_i^*)}\right)=0,\quad i=1,\cdots,n,$$
$$\nu^*\geqslant\frac{1}{(\alpha_i+x_i^*)},\quad i=1,\cdots,n.$$

如果$\nu^*<\dfrac{1}{\alpha_i}$，只有当$x_i^*>0$时最后一个条件才成立。而由第三个条件可知$\nu^*=\dfrac{1}{(\alpha_i+x_i^*)}$。求解$x_i^*$，我们得出结论：当$\nu^*<\dfrac{1}{\alpha_i}$时，有$x_i^*=\dfrac{1}{\nu^*}-\alpha_i$。如果$\nu^*\geqslant\dfrac{1}{\alpha_i}$，那么$x_i^*>0$不会发生。这是因为如果$x_i^*>0$，那么$\nu^*\geqslant\dfrac{1}{\alpha_i}>\dfrac{1}{(\alpha_i+x_i^*)}$，

违背了互补松弛条件。因此，如果$\nu^* \geq \dfrac{1}{\alpha_i}$，那么$x_i^* = 0$。所以有下式：

$$x_i^* = \begin{cases} \dfrac{1}{\nu^*} - \alpha_i, & \nu^* < \dfrac{1}{\alpha_i} \\ 0, & \nu^* \geq \dfrac{1}{\alpha_i} \end{cases}$$

。或者，更简洁地，$x_i^* = \max\left\{0, \dfrac{1}{\nu^*} - \alpha_i\right\}$。将$x_i^*$的

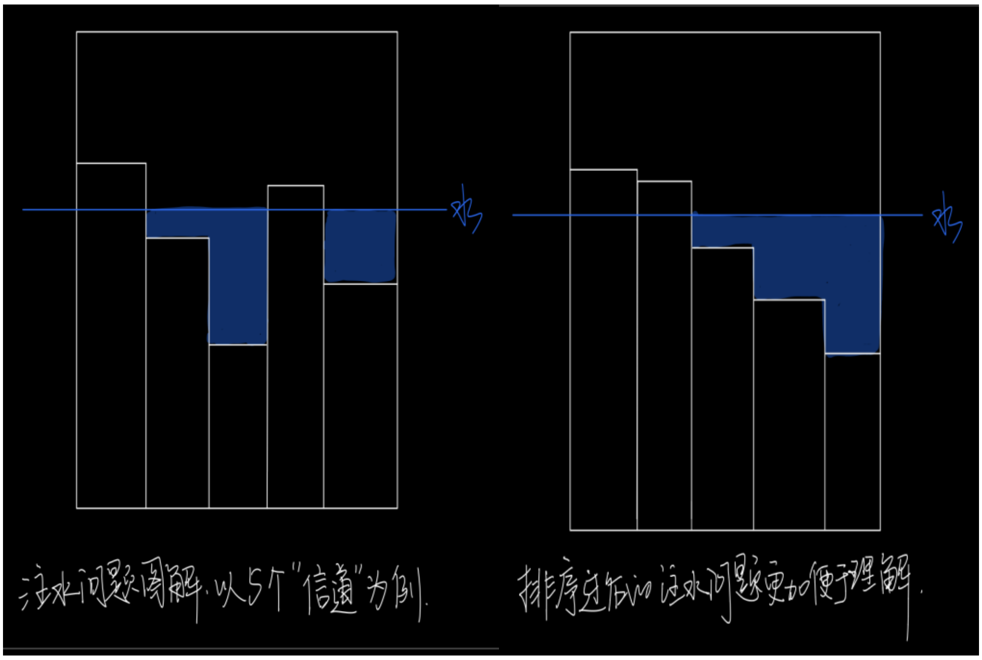
表达式代入条件$1^T x^* = 1$我们得到：$\sum_{i=1}^{n} \max\left\{0, \dfrac{1}{\nu^*} - \alpha_i\right\} = 1$。

方程左端是$\dfrac{1}{\nu^*}$的分段线性增函数，分割点为α_i，因此上述方程有唯一确定的解。

本次实验即通过 *python* 编写程序，计算注水问题的最优值。

3. 分析及算法过程

考虑注水的过程，如下图。



算法过程：

1) 将生成的α进行从小到大的排序

```python
''' sort the ndarray  (major point)'''
alpha = np.sort(alpha,0)
```

2) 注水：

① 首先生成与α维数相同的 x_array 集（类型：ndarray），分别存放对应每组将会注入的水量。

② 对α进行遍历。计算第 i 组和第 i+1 组之间的高度差。(要先将高度差的部分注满水，才能让第 i 组和第 i+1 组的水量一起上涨。)

③ I. 如果余下的水量分给前 i 组之后，使这 i 组水面上涨的高度大于第 i 组和第 i+1 组之间的高度差，证明在这一次循环中水量很充足，我们要做的就是把高度差的部分补满，使前 i 组和第 i+1 组的高度为一样的，同时减去消耗的水量。

II. 如果余下的水量给前 i 组之后，使这 i 组水面上涨的高度小于第 i 组和第 i+1 组之间的高度差，证明在这一次循环中水量不够了，我们要做的是把上涨的高度求出来，加上第 i 组原有的高度（因为第 i 组此时还未注水），所得到的高度就是 Horizontal_line。

III. 如果把所有的组都遍历过了，则考虑α的界限是[0,1]，用 1 构造一个“不存在的”组，再次进行上述步骤，此时一定能得到 Horizontal_line。

注：如果水量一直充足，要考虑到生成维数的问题，将 Horizontal_line 的值设为 1，同时打印文字提示此次生成的数据不合适。

④ 再次对α进行遍历，此次遍历时通过 Horizontal_line 与α中每一个元素的比较来得出 x_array 中元素的值。如果 Horizontal_line 比α_i大，证明有水注入，此时 x_array[i]=Horizontal_line-α_i；如果 Horizontal_line 比α_i小，证明没有水注入，此时 x_array[i]=0。

```python
def fill_water(alpha, total_water, precision):  # water filling algorithm
    x_array = np.zeros((dimension,1),dtype='float_')
    left_water=total_water
    temp2=0.0
    for i in range (dimension):
        if(i!=dimension-1):
            temp=alpha[i+1]-alpha[i]
            if(left_water/(i+1)<=temp):
                temp2=left_water/(i+1)+alpha[i]    # find the horizontal line
                break
            else:
                left_water-=temp*(i+1)
        else:
            temp = 1 - alpha[i]
            if (left_water / (i + 1) <= temp):
                temp2 = left_water / (i + 1) + alpha[i]  # find the horizontal line
                break
            else:
                print('！！！！！题目生成错误，需要的水量小于1，一定能注满！！！！！')    # 异常处理
                temp2 = 1
    for i in range(dimension):
        if(temp2 - alpha[i]>=0):
            x_array[i]=temp2-alpha[i]
        else:
            x_array[i]=0
    return x_array
```

终止条件

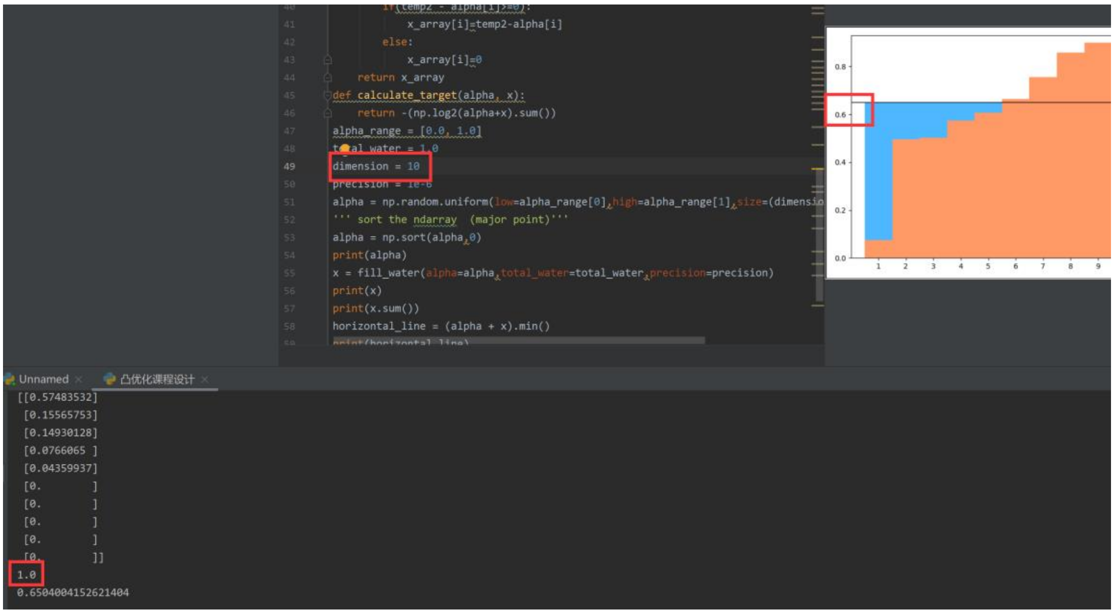
3) 程序的可视化

```python
def visualize_water(alpha, x, horizontal_line): # 可视化
    alpha = alpha.squeeze()
    x = x.squeeze()
    x_range = range(1, x.shape[0]+1)
    plt.xticks(x_range)
    plt.bar(x_range, alpha, color='#ff9966',
    width=1.0, edgecolor='#ff9966')
    plt.bar(x_range, x, bottom=alpha, color='#4db8ff', width=1.0)
    plt.axhline(y=horizontal_line,linewidth=1, color='k')
    plt.show()
```

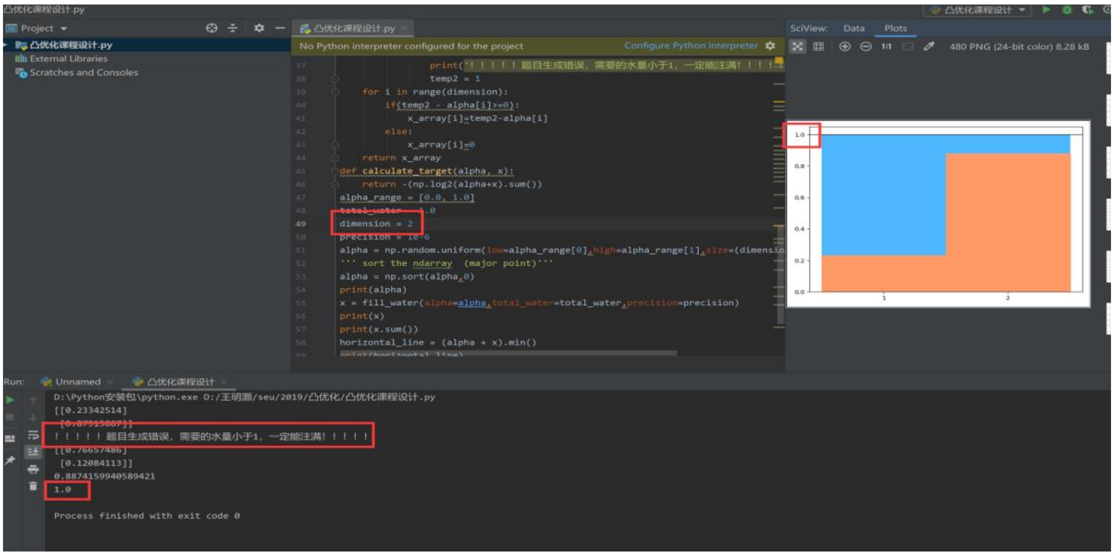
4. 实验结果截图及分析：

1）实验结果截图

① 正确注水的截图

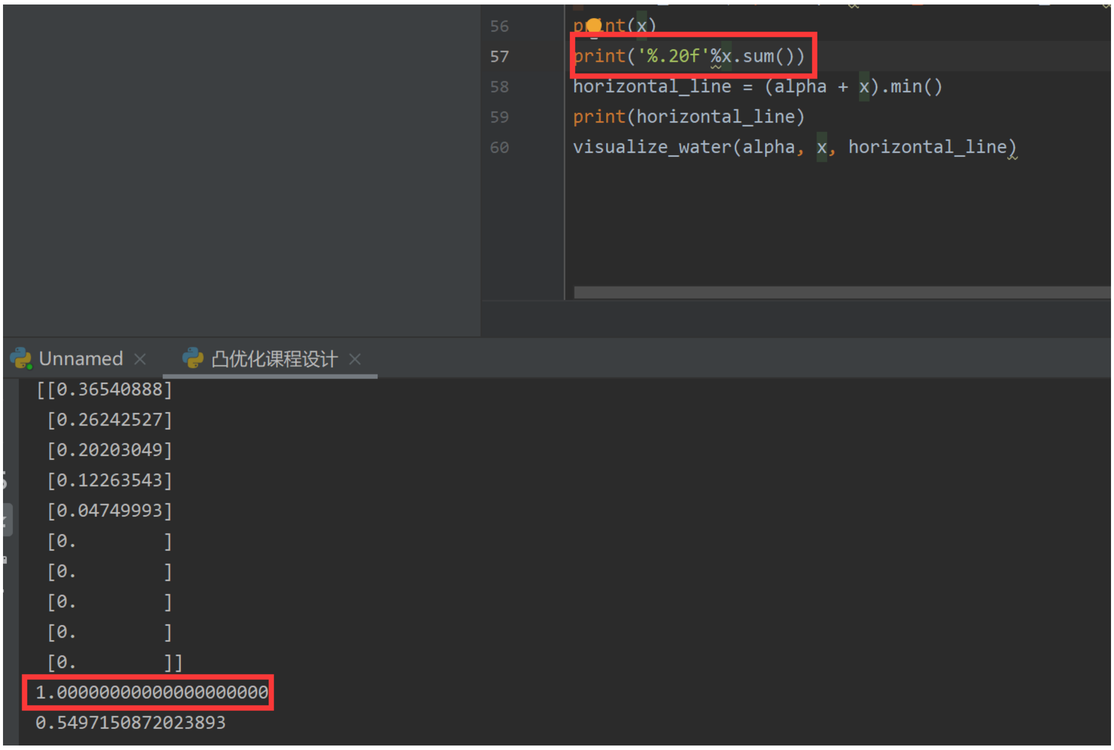


② 注满了的情况

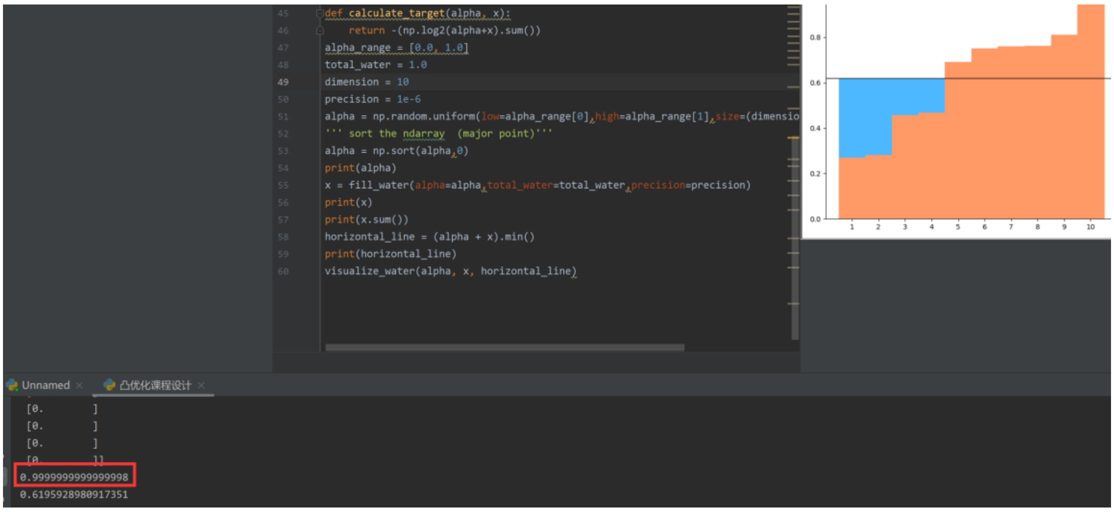


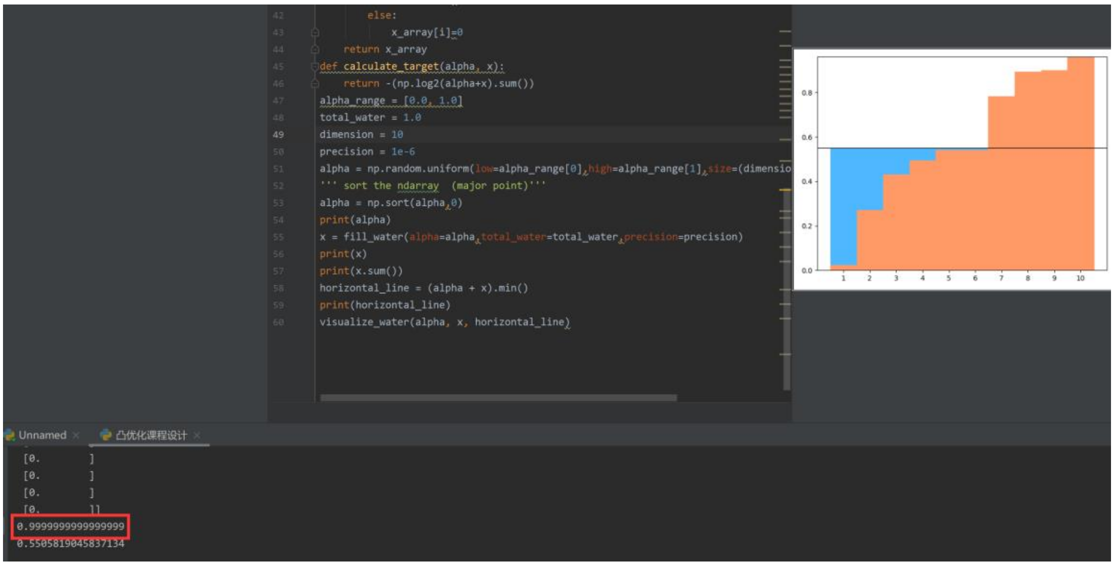
2）结果分析：

在得到结果之后，与同学进行一些交流，发现大家的 x.sum()精度都是小数点后有若干位(如 1.00234567323 之类)，考虑过使用格式化使输出强制增加为 20 位小数，然而结果是 1.00000000000000000000。



转变思路，多次进行实验，发现实验比较“失败”的情况下 x.sum()输出结果为 0.9999999999999or0.99999999998,大部分情况下为 1.0。了解到很多同学用二分法进行计算，遂不要脸的认为此算法比较优秀，精确度特别高。





①: In an optimization problem, a **slack variable** is a variable that is added to an inequality constraint to transform it into an equality. Introducing a slack variable replaces an inequality constraint with an equality constraint and a non-negativity constraint on the slack variable. (Wikipedia)