HW3-Nonlinear Programming

Promotion Problem

Analysis

- Decision Variable
  - 两种商品的促销水平，记为x_j

$$x_j \quad j=1,2$$

- Objective Function
  - 两种商品的总销售收入，记为z

$$max \quad z=\sum_{j=1}^{2} f(x_j)=3x_1-(x_1-1)^2+3x_2-(x_2-2)^2$$

- Constraints
  - 显性约束
    1. 促销水平受资源限制

$$\begin{aligned} 4x_1+x_2 &\leq 20 \\ x_1+4x_2 &\leq 20 \end{aligned}$$

  - 隐性约束
    1. 促销水平不会为负

$$\begin{aligned} x_1 &\geq 0 \\ x_2 &\geq 0 \end{aligned}$$

Model

$$max \quad z=\sum_{j=1}^{2} f(x_j)=3x_1-(x_1-1)^2+3x_2-(x_2-2)^2$$

$$s.t. \begin{cases} 4x_1+x_2 \leq 20 \\ x_1+4x_2 \leq 20 \\ x_1 \geq 0 \\ x_2 \geq 0 \end{cases}$$

Solve

由于PuLP无法求解非线性规划问题，我这次换用了SciPy的optimize模块（也算是学点新知识）

- Model

```
obj = lambda x: -(3*x[0] - (x[0]-1)**2 + 3*x[1] - (x[1]-2)**2)
bnds = ((0,None),(0,None))
cons = ({'type': 'ineq', 'fun': lambda x: 4 * x[0] + x[1] - 20},
        {'type': 'ineq', 'fun': lambda x: 4 * x[1] + x[0] - 20})
```

- Result
  - State

```
     fun: -10.999999999999998
     jac: array([3., 1.])
 message: 'Optimization terminated successfully'
    nfev: 10
     nit: 3
    njev: 3
  status: 0
 success: True
       x: array([4., 4.])

Process finished with exit code 0
```

- Output

x1	x2	Obj
4.0	4.0	10.99

Producing Problem

Analysis

- Mathematical Description
  - 记第i轮获得的总利益为z_j

    Round1: $z_1 = g(y_1) + h(x_1 - y_1) \quad x_1 = x_1$

    Round2: $z_2 = g(y_2) + h(x_2 - y_2) \quad x_2 = ay_1 + b(x_1 - y_1)$

    Round2: $z_3 = g(y_3) + h(x_3 - y_3) \quad x_3 = ay_2 + b(x_2 - y_2)$

    ……

- Decision Variable
  - 第j轮生产时，投入产品A的原料数

$$y_j \quad j = 1, 2, .., n$$

- Objective Function
  - n轮生产的总利润，记为z

$$max \quad z = \sum_{j=1}^{n} z_j$$

$$z_j = g(y_j) + h(x_j - y_j)$$

这个公式展开的话会比较复杂，反而是分离形式计算机迭代起来轻松，因此就不写出展开形式了

- Constraints
  - 显性约束
    1. 回收率规则

$$x_j = ay_{j-1} + b(x_{j-1} - y_{j-1})$$

  - 隐性约束
    1. 每次生产A的数量大于等于0，但小于回收剩下的最大额度

$$0 \le y_j \le x_j$$

$$x_j = ay_{j-1} + b(x_{j-1} - y_{j-1})$$

Model

$$max \quad z = \sum_{j=1}^{n} z_j$$

$$s.t. \begin{cases} 0 \le y_j \le x_j \\ x_j = a y_{j-1} + b(x_{j-1} - y_{j-1}) \\ z_j = g(y_j) + h(x_j - y_j) \end{cases}$$

这是个多步骤的决策，虽然感觉上更像一个动态规划问题，但其实写出来之后也可以发现，由于规则明确简单，直接在y空间中进行决策就是可行的

附录

Prob1 Code

```
import numpy as np
import sympy as sym
from scipy.optimize import minimize

def objective_function():
    obj = lambda x: -(3*x[0] - (x[0]-1)**2 + 3*x[1] - (x[1]-2)**2)
    return obj

def constrains():
    cons = ({'type': 'ineq', 'fun': lambda x: 4 * x[0] + x[1] - 20},
        {'type': 'ineq', 'fun': lambda x: 4 * x[1] + x[0] - 20})
    return cons

if __name__ == "__main__":
    ## 初始迭代点
    x0 = np.array([1,1])
    ## 约束
    cons = constrains()
    bnds = ((0,None),(0,None))
    res = minimize(objective_function(), x0, bounds=bnds, constraints=cons)
    print(res)
```