最优化方法课程作业
系统使用说明

学院：信息科学与工程学院
班级：计算机技术
学号：2020317041
姓名：王栋

2020 年 12 月 6 日

目录

系统说明文档

任务介绍

单纯形法是求解线性规划问题最常用、最有效的算法之一。单纯形法最早由 George Dantzig 于 1947 年提出，近 70 年来，虽有许多变形体已经开发，但却保持着同样的基本观念。如果线性规划问题的最优解存在，则一定可以在其可行区域的顶点中找到。基于此，单纯形法的基本思路是：先找出可行域的一个顶点，据一定规则判断其是否最优；若否，则转换到与之相邻的另一顶点，并使目标函数值更优；如此下去，直到找到某最优解为止。

运输问题是特殊的线性规划问题，因此有他特殊的求解方法。如果使用线性规划去求解，因为运输问题的变量比较多将会出现大量的退化现象。所以用单纯型法去求解计算的时间会比较多一点。当然运输问题作为一个特殊的线性规划问题，人们研究出了求解方法。利用单纯型的思想用到运输问题里面，就得到了运输问题的表上作业法。

本系统解决了以上两类问题，一是利用单纯形法解决线性规划求解最小值问题，二是使用表上作业法解决运输问题。

系统介绍

该系统使用 Python 语言编写，借助于 Python 丰富的函数库实现，主要使用了 Numpy 以及 TKinter 实现。
NumPy（Numerical Python）是 Python 的一种开源的数值计算扩展。这种工具可用来存储和处理大型矩阵，比 Python 自身的嵌套列表（nested list structure)结构要高效的多，支持大量的维度数组与矩阵运算，此外也针对数组运算提供大量的数学函数库。使用 Numpy 能够简化数据运算操作，使用更少的代码量来实现特定的功能。

Python 同时也提供了许多 gui 框架，本系统一开始希望使用 easygui 实现，然而 easygui 过于简洁，不能满足系统需求，遂使用 Tkinter 编写界面。Tkinter 模块(Tk 接口)是 Python 的标准 Tk GUI 工具包的接口.Tk 和 Tkinter 可以在大多数的 Unix 平台下使用，同样可以应用在 Windows 和 Macintosh 系统里。Tk8.0 的后续版本可以实现本地窗口风格，并良好地运行在绝大多数平台中。

系统实现

本系统可分为单纯形法求解、运输问题求解与交互界面三部分。

单纯形法实现

单纯形法实现参考自《最优化理论与算法》第三章中单纯形方法计算步骤，具体描述如下：

3.1.2 单纯形方法计算步骤

我们以极小化问题为例给出计算步骤. 首先要给定一个初始基本可行解. 设初始基为 $\boldsymbol{B}$，然后执行下列主要步骤：

(1) 解 $\boldsymbol{B}\boldsymbol{x}_B=\boldsymbol{b}$，求得 $\boldsymbol{x}_B=\boldsymbol{B}^{-1}\boldsymbol{b}=\bar{\boldsymbol{b}}$，令 $\boldsymbol{x}_N=\boldsymbol{0}$，计算目标函数值 $f=\boldsymbol{c}_B\boldsymbol{x}_B$.

(2) 求单纯形乘子 $\boldsymbol{w}$，解 $\boldsymbol{w}\boldsymbol{B}=\boldsymbol{c}_B$，得到 $\boldsymbol{w}=\boldsymbol{c}_B\boldsymbol{B}^{-1}$. 对于所有非基变量，计算判别数 $z_j-c_j=\boldsymbol{w}\boldsymbol{p}_j-c_j$. 令

$$z_k-c_k=\max_{j\in R}\{z_j-c_j\}.$$

若 $z_k-c_k\leqslant 0$，则对于所有非基变量 $z_j-c_j\leqslant 0$，对应基变量的判别数总是零，因此停止计算，现行基本可行解是最优解. 否则，进行下一步.

(3) 解 $\boldsymbol{B}\boldsymbol{y}_k=\boldsymbol{p}_k$，得到 $\boldsymbol{y}_k=\boldsymbol{B}^{-1}\boldsymbol{p}_k$，若 $\boldsymbol{y}_k\leqslant\boldsymbol{0}$，即 $\boldsymbol{y}_k$ 的每个分量均非正数，则停止计算，问题不存在有限最优解. 否则，进行步骤(4).

(4) 确定下标 r，使

$$\frac{\bar{b}_r}{y_{rk}}=\min\left\{\frac{\bar{b}_i}{y_{ik}}\middle|y_{ik}>0\right\},$$

x_{B_r} 为离基变量，x_k 为进基变量. 用 $\boldsymbol{p}_k$ 替换 $\boldsymbol{p}_{B_r}$，得到新的基矩阵 $\boldsymbol{B}$，返回步骤(1).

算法实现如下所示

```
import numpy as np
import decimal
obj = np.array([-4, -1, 0, 0, 0])
st = np.array([[-1, 2, 1, 0, 0, 4],[2, 3, 0, 1, 0, 12],[1, -
1, 0, 0, 1, 3]])

def simplex(obj, st):
    row, column = st.shape
    A = np.delete(st, column-1, 1)
    b = st[:,column-1]
    p = np.array([example for example in A.T])
    # 取后几个数字的索引
    index = [e for e in range(len(p))][-row:]
    while(True):
        B = p[index].T

        # B 矩阵求逆
```

```
        B_inv = np.linalg.inv(B)
        xb = np.matmul(B_inv,b)

        # 得到的一个目标值
        cb = obj[index]
        # 使用近似值，解决浮点运算不准确的问题
        f = np.around(cb@xb)

        # 单纯形乘子
        w = cb@B_inv

        # 计算判别数，存储在字典中
        judge = {}
        for i,_ in enumerate(p):
            if i not in index:
                judge[i] = w@p[i]-obj[i]

        # 所有判别数都小于 0，停止迭代
        if(max(judge.values())<=0):
            print("找到最优解的值为{}".format(f))
            return f
#         else:
#             print("继续迭代")

        # 获取最大值的索引，xk 为进基变量
        k = max(judge, key=judge.get)

        yk = B_inv@p[k]

        # 判断 yk>0 中是否有 True。
        bo = False
        for e in yk>0:
            bo = bo|e
        if(not bo):
            print("该问题不存在最优解")
            return np.nan

        # 选择下标 r，xr 为离基变量
        dic = {}
        for i,_ in enumerate(yk):
            if(yk[i]>0):
                dic[i]=b[i]/yk[i]
        r = min(dic, key=dic.get)
        # 由 index 确定 p 变量
```

```
61.         index[r] = k
```

实现该部分时也遇到了一些小问题，在此讲述出现的问题以及解决方法：

1. 对于每次选取的列向量（如$p1\ p2\ p4$），算法实现中不好描述，因此将全部向量构造成一个列表，同时构建$index$列表表示每次选择的变量序号。
2. 在判断是否存在最优解时，需判断yk中是否存在大于 0 的数值。用$if\ True\ in\ yk \geq 0$ 会产生逻辑错误，具体原因没有探究。采用的办法是将$yk \geq 0$列表中所有布尔值相或，这样的话若存在$Ture$最后的结果也为$True$，否则为$False$。个人认为是一种比较奇妙的解决方法。
3. 由于计算机使用了二进制存储，浮点数在计算机中为近似估计值，对于浮点数的运算会产生精度问题，比如本样例中正确结果为 18，却会输出-18.000000000000004。对于数值计算无需特别精确，使用简单的四舍五入解决此类问题。

运输问题实现

算法实现如下所示：

```
1.  import numpy as np
2.  from collections import Counter
3.  import warnings
4.  warnings.filterwarnings("ignore")
5.
6.  def find_initial_solution(costs, demand, supply):
7.      C = np.copy(costs)
8.      d = np.copy(demand)
9.      s = np.copy(supply)
10.
11.     # Get the shape of costs-matrix
12.     n, m = C.shape
13.
14.     # Create the matrix of basic values and convert cost to one-dim array
15.     X = np.zeros((n, m))
16.     indices = [(i, j) for i in range(n) for j in range(m)]
17.     xs = sorted(zip(indices, C.flatten()), key=lambda kv: kv[1])
18.
19.     # Find initial solution
20.     for (i, j), _ in xs:
21.         if d[j] == 0:
22.             continue
23.         else:
24.             # Reserving supplies in a greedy way
25.             remains = s[i] - d[j] if s[i] >= d[j] else 0
26.             grabbed = s[i] - remains
```

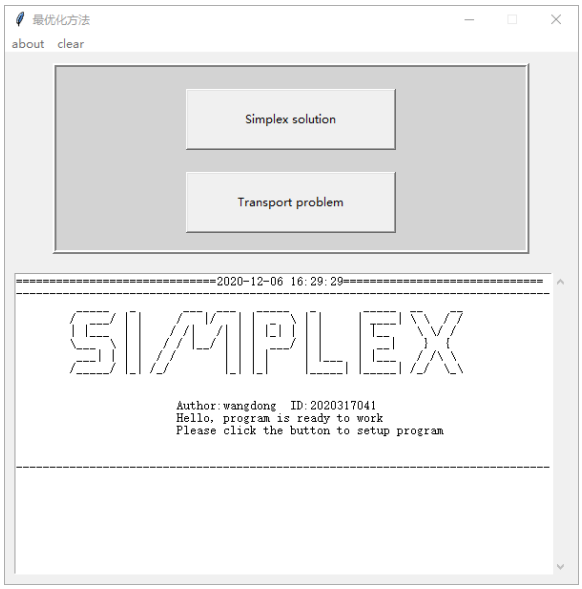
```
            X[i, j] = grabbed
            s[i] = remains
            d[j] -= grabbed
    return X


def find_potential(X, C):
    n, m = X.shape

    u = np.array([np.nan] * n)
    v = np.array([np.nan] * m)

    _x, _y = np.where(X > 0)
    nonzero = list(zip(_x, _y))
    f = nonzero[0][0]
    u[f] = 0

    while any(np.isnan(u)) or any(np.isnan(v)):
        for i, j in nonzero:
            if np.isnan(u[i]) and not np.isnan(v[j]):
                u[i] = C[i, j] - v[j]
            elif not np.isnan(u[i]) and np.isnan(v[j]):
                v[j] = C[i, j] - u[i]
            else:
                continue
    return u, v


def transport(costs, demand, supply):
    # Get initials solution
    n, m = costs.shape
    X = find_initial_solution(costs, demand, supply)

    # 打印在控制台上的值
    print("基本可行解:", np.sum(X * costs))

    while True:
        S = np.zeros((n, m))

        # Find potentials
        u, v = find_potential(X, costs)

        # Find S - matrix
        for i in range(n):
```

```
            for j in range(m):
                S[i, j] = costs[i, j] - u[i] - v[j]

        # Condition to break
        s = np.min(S)
        if s >= 0:
            print("求得的最优解为：", np.sum(X * costs))
            break

        i, j = np.argwhere(S == s)[0]
        start = (i, j)

        # print(start)
        # Find cycle elements

        T = np.copy(X)
        T[start] = 1
        while True:
            _xs, _ys = np.nonzero(T)
            xcount, ycount = Counter(_xs), Counter(_ys)

            for x, count in xcount.items():
                if count <= 1:
                    T[x, :] = 0
            for y, count in ycount.items():
                if count <= 1:
                    T[:, y] = 0

            if all(x > 1 for x in xcount.values()) \
                    and all(y > 1 for y in ycount.values()):
                break
        # print(T)

        # Finding cycle order
        dist = lambda kv1, kv2: abs(kv1[0] - kv2[0]) + abs(kv1[1] - kv2[1])

        fringe = [tuple(p) for p in np.argwhere(T > 0)]
        # print(fringe)

        size = len(fringe)

        path = [start]
        while len(path) < size:
            last = path[-1]
```

```
            if last in fringe:
                fringe.remove(last)
            next = min(fringe, key=lambda kv: dist(last, (kv[0], kv[1])))
            path.append(next)

        # Improving solution on cycle elements
        neg = path[1::2]
        pos = path[::2]
        q = min(X[list(zip(*neg))])

        # Print optimal solution
        if q == 0:
            print("基本可行解:", np.sum(X * costs))

        # Improve solution
        X[list(zip(*neg))] -= q
        X[list(zip(*pos))] += q

        # Print table after improving
        print("基本可行解:", np.sum(X * costs))

```

交互界面实现

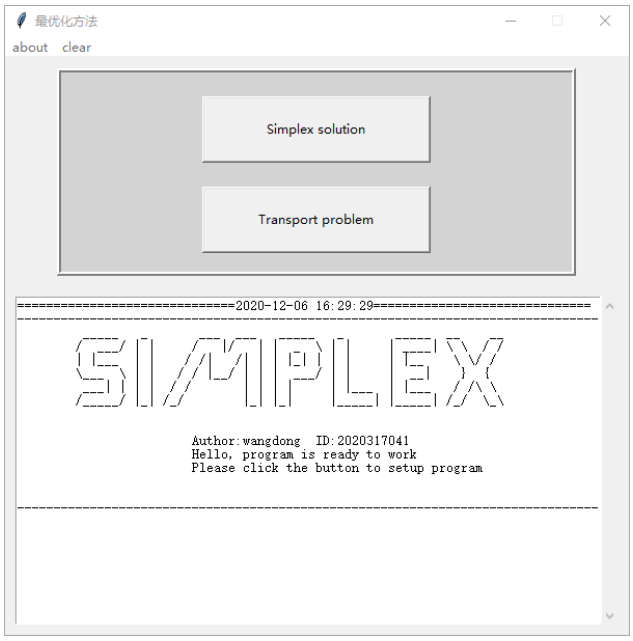
本部分借助于 Python 函数库中的 TKinter 模块实现，主要使用了 Button、Label、ScrollText 控件，主界面如下：



GUI 的具体实现与本课程无关，因此在后面章节中重点介绍使用方法。

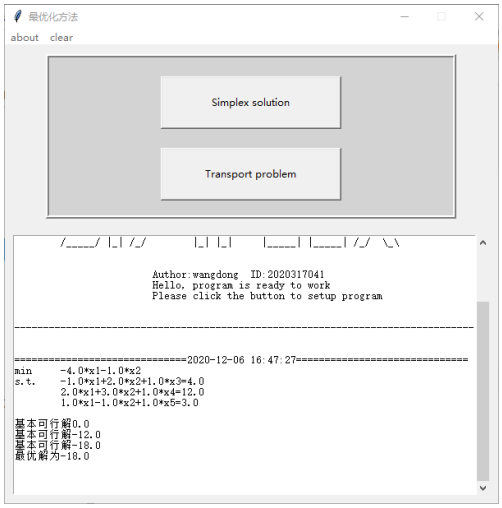
使用教程

该系统可采用两种方式运行，一是从 ipynb 源码运行，按照顺序运行代码后会调出系统主界面；而是直接运行 exe 文件。下面对于该系统的界面以及如何操作界面进行介绍。



如图所示为主界面，主要包括两个区域，控制区与输出区（模拟控制台），控制区包含两个按钮，分别表示求解两类问题，点击后会弹出相应的窗口。输出区会输出运算结果，以及一些简单的文字以丰富界面。点击菜单栏中 clear 会清空输出区域，点击 about 会弹出一个介绍窗口。

点击按钮后会出现如图所示的界面，在输入框中输入系数矩阵后，点击输入完成会关闭此弹出框，完成相应的运算并将运算结果输出到主界面输出区域。



缺陷和不足

由于时间紧张与个人技术原因，本系统仅实现了主要功能，距离完整的系统还存在着许多缺陷与不足。

1. 输入只能输入系数矩阵，可能会影响用户体验。
2. 对于输入没有进行校验，若输入不正确可能会出现未知的错误，在健壮性表现上不太好。
3. 算法运行时，没有输出具体的求解步骤。

参考资料

1. 《最优化理论与算法（第 2 版）》 陈宝林 最优化理论与算法 (豆瓣) (douban.com)
2. https://github.com/BON4/TransportationProblem
3. https://baike.baidu.com/item/%E5%8D%95%E7%BA%AF%E5%BD%A2%E6%B3%95/8580570?fr=aladdin
4. https://baike.baidu.com/item/%E8%BF%90%E8%BE%93%E9%97%AE%E9%A2%98/12734790?fr=aladdin
5. https://blog.csdn.net/python1212/article/details/101421435