云南大学数学与统计学院
《运筹学通论实验》上机实践报告

课程名称：运筹学实验	**年级：**2015 级	**上机实践成绩：**
指导教师：李建平	**姓名：**刘鹏	**专业：**信息与计算科学
上机实践名称：两阶段法求线性规划问题	**学号：**20151910042	**上机实践日期：**2018-07-07
上机实践编号：3	**组号：**	

一、 实验目的

通过对两阶段法进行编程实现，让自己对单纯形算法理解得更加透彻；

通过对 MATLAB 的 `linprog` 程序进行调用，学习使用 MATLAB 的优化功能。

二、 实验内容

写出两阶段法[1]的算法；

用 C 语言[2]编程实现两阶段算法。

三、 实验平台

Microsoft Windows 10 Pro Workstation 1803；

MathWorks MATLAB R2018a；

Microsoft Visual Studio 2017 Enterprise.

四、 算法设计 [1]

单纯形算法是一个迭代方法，在每次迭代中，我们的目标是重新整理线性规划，使得基本解有一个更大的目标值。我们选择一个在目标函数中系数为正的非基本变量x_e，而且尽可能增加x_e的值而不违反任何约束。变量x_e称为基本变量，并且某个其他变量x_l变成非基本变量。

Algorithm **SIMPLEX**, Simplex Method for solving LP problems can start with 0.

Input (1) 系数矩阵$\boldsymbol{A} = (a_{ij})_{m\times n}$，$\boldsymbol{A} = (\boldsymbol{A}_1, \boldsymbol{A}_2, \cdots, \boldsymbol{A}_m)$，$\boldsymbol{A}_i$是系数矩阵的第$i$列;

(2) 价值向量$\boldsymbol{c} = (c_{ij})_{n\times 1} = (c_1, c_2, ,\cdots, c_n)$;

(3) 常数向量$\boldsymbol{b} = (b_{ij})_{m\times 1} = (b_1, b_2, \cdots, b_m) \geq \boldsymbol{0}$;

It means to find $\mathrm{MAX}(\boldsymbol{cx})$, s.t. $\boldsymbol{Ax} = \boldsymbol{b}$, and $\boldsymbol{x} \geq \boldsymbol{0}$.

Output 如果有最优解，输出最优解$\boldsymbol{x}$，如果没有，输出 No Solution

Begin

1 此处的伪代码中，矩阵运算符的意义均与 MATLAB 语言一致，如矩阵的左除、右除和点除等。

Step 1 $\boldsymbol{A}' = (-\boldsymbol{c}^{\mathrm{T}},\ \boldsymbol{A}^{\mathrm{T}})^{\mathrm{T}} = (a'_{ij})_{(m+1)\times n}$;

Step 2 $\boldsymbol{b}' = (0,\ \boldsymbol{b}^{\mathrm{T}})^{\mathrm{T}} = ((b'{}_{ij})_{(m+1)\times 1})$;

Step 3 $\boldsymbol{A}'' = (\boldsymbol{A}', \boldsymbol{b}') = (a''{}_{ij})_{(m+1)\times(n+1)}$;

Step 4 记$\boldsymbol{A}''$的第一行为$\boldsymbol{A}_0$;

Step 5 从$-\boldsymbol{A}_0$中价值元素中的找寻最大的正数，命之为Pivot，记之在此行中的坐标为C，GOTO **Step 6**；如果经过标记之后回到这里而且找不到正数，说明循环结束，输出$z = \boldsymbol{A}''(1, n+1)$，$x = \boldsymbol{A}''(:, n+1)^{\mathbf{T}}$，GOTO **End**.
如果在所有价值元素中找不到正数，说明这可能是通过变化 MIN 类型的价值向量得来的，如果原先的约束都是小于等于，那么毫无疑问**0**就是最优解，GOTO **End**；如果原先的约束有大于等于约束与等于约束或者其中之一，那么很显然**0**并不行，因为它要求松弛变量与剩余变量的最终解都为0，这是荒谬的，也就是说**0**并不行，还需要进行如下处理：在矩阵的第一行中，消去所有的非零变量，即通过基本行变换将那些负数变为0，这样更新过的元素中就有正数了（因为变零的操作是加法，如果所有变换都产生不了正数，那说明只有**0**这个不符合题意的解能单纯地满足约束方程），表现为$\boldsymbol{A}''$的第一行有负数，$\mathrm{MIN}(\boldsymbol{A}_0) < 0$，从$-\boldsymbol{A}_0$中价值元素中的找寻最大的正数，命之为Pivot，记之在此行中的坐标为C，设置本次迭代不在经过此步骤，否则会有死循环，GOTO **Step 6**;

Step 6

利用条件除法作集合$\boldsymbol{S} = \{b_i/a_{i,C} \mid a_{i,C} > 0, \mathrm{i} = 2, 3, \cdots,\ m+1\}$，$t = \mathrm{MAX}(\boldsymbol{S})$，最大值不止一个就选其中一个，记$t$在$\boldsymbol{A}''$中的坐标为$(R,\ C)$，GOTO **Step 7**

Step 7

$\boldsymbol{A}''(R,:) = \boldsymbol{A}''(R,:)/\boldsymbol{A}''(R,C)$;
for i through 1 to $m+1$
  if $i \neq R$, then let $\boldsymbol{A}''(i,:) = \boldsymbol{A}''(i,:) - \boldsymbol{A}''(i,C)/\boldsymbol{A}''(R,C)$
Go to **Step 5**;

End

下面是调用了上面的单纯形算法的两阶段算法。

Algorithm **DUAL-SIMPLEX**, 重点解决初值不能从**0**开始的 LP 问题

Input (1) 系数矩阵$\boldsymbol{A} = (a_{ij})_{m\times n}$，$\boldsymbol{A} = (\boldsymbol{A}_1, \boldsymbol{A}_2, \cdots, \boldsymbol{A}_m)$，$\boldsymbol{A}_i$是系数矩阵的第$i$列;
(2) 价值向量$\boldsymbol{c_1} = (c_{ij})_{n\times 1} = (c_1,\ c_2,\ ,\cdots,\ c_n)$;
(3) 价值向量$\boldsymbol{c_2} = (c_{ij})_{n\times 1} = (c_1,\ c_2,\ ,\cdots,\ c_n)$;
(4) 常数向量$\boldsymbol{b} = (b_{ij})_{m\times 1} = (b_1,\ b_2,\ \cdots,b_m) \geq \mathbf{0}$;

Output 如果有最优解，输出最优解$\boldsymbol{x}$，如果没有，输出 No Solution

Begin

Step 1 **SIMPLEX**$(\boldsymbol{A},\ \boldsymbol{c_1},\ \boldsymbol{b})$，输出$z = \boldsymbol{A}''(1, n+1)$，此时如果$z = 0$，GOTO **Step 2**，若不为零，则输出无解，GOTO **End**;

Step 2 重新初始化单纯形表以为第二阶段做准备：把$\boldsymbol{A}''$中人工变量所在的行全部删除或者全部变为零，把第一行用$(-\boldsymbol{c_2},\ 0)$代替；GOTO **Step 3**;

Step 3 将$\boldsymbol{A}''$传给 **SIMPLEX** 算法的 **Step 4**，完成计算。

End

五、 程序代码

5.1 程序描述

为了完成此项目，我创建了很多新的类，有读取类，动态数组类，矩阵类等，因为是针对单纯形算法进行量身打造，所以拓展性一般，在后来的几个新的项目里，我又在稍加修改的基础上复用了这几个类，并且针对其中几个可以通用的类进行了优化，比如加深抽象层次使之可以作为泛型容器。回过头来再看原先的代码并不足够优秀。但是时间有限，提交在即，这里就不再进行代码优化升级或者某些地方的重构了，毕竟这个程序做了大量的测试，并没有致命的 bug，特此说明。

因为程序很长，文件很多，这里就仅仅列出核心文件 Simplex.h 中的代码，其余的代码在附录中给出。本来头文件里不应该写实现，但是这里的程序不需要封装，所以实现与定义放在一起呈现。

5.2 程序代码

```
/*
* Copyright (c) 2018, Liu Peng, School of Mathematics and Statistics, YNU
* Apache License.
*
* 文件名称：Simplex.h
* 文件标识：见配置管理计划书
* 摘　要：对标准输入的单纯形问题进行求解
*
* 当前版本：1.0
* 作　者：刘鹏
* 创建日期：2018 年 5 月 4 日
* 完成日期：2018 年 5 月
*
* 取代版本：
* 原作者 ：刘鹏
* 完成日期：
*/

#pragma once
```

```c

#include "Matrix_Operation.h"
#include "Divide.h"

// Generally speaking, this data structure is not a table.
// whatever, it works.
typedef struct Simplex_Tableau {
    Matrix *Matrix;
    Dynamic_Array *Objective_Vector;
    Dynamic_Array *b;
} Simplex_Tableau;

// Initialize the table of simplex method.
// This is a simple implementation, only can solve problems like "Ax = b"
// with all the slack variables has been added.
Simplex_Tableau *Simplex_Tableau_init(char *c, char *A, char *b) {
    Simplex_Tableau *ans = (Simplex_Tableau *)calloc(1, sizeof(Simplex_Tableau));

    Matrix *m = get_Matrix(A);
    ans->Objective_Vector = get_Dynamic_Array(c);
    ans->b = get_Dynamic_Array(b);

    int i = 0;
    int j = 0;
    Dynamic_Array *tmp = Dynamic_Array_init();

    // STEP 1: Append the zero'th row.
    for (i = 1; i <= ans->Objective_Vector->n; i++) {
        Dynamic_Array_append(tmp, -1 * Dynamic_Array_get_Element(ans->Objective_Vector, i));
    }
    Dynamic_Array_append(tmp, 0);

    // STEP 2: Append the Coefficient Matrix.
    for (i = 1; i <= m->n_row; i++) {
        for (j = 1; j <= m->n_column; j++) {
            Dynamic_Array_append(tmp, Matrix_get_Element(m, i, j));
        }
        Dynamic_Array_append(tmp, Dynamic_Array_get_Element(ans->b, i));
    }

    ans->Matrix = Matrix_init(m->n_row + 1, m->n_column + 1);
    ans->Matrix->low_level_array = tmp->A;

    return ans;
}

// c2 is needed in the second phase.
// The Objective Vector need to be changed. reshape the matrix.
```

```c
Simplex_Tableau *Simplex_Tableau_re_init(Simplex_Tableau *S, char *c2) {
    int i = 1;
    for (; i <= S->Objective_Vector->n; i++) {
        double tmp = Dynamic_Array_get_Element(S->Objective_Vector, i);
        if (tmp > 1e-15 || tmp < -1e-15) {
            Matrix_column_to_zero(S->Matrix, i);
        }
    }

    Dynamic_Array *New_Objective_Vector = get_Dynamic_Array(c2);
    S->Objective_Vector = New_Objective_Vector;

    for (i = 0; i < New_Objective_Vector->n; i++) {
        // Cover the old value
        *(S->Matrix->low_level_array + i) = Dynamic_Array_get_Element(New_Objective_Vector, i + 1);
    }
    Matrix_num_mul_vector(-1, S->Matrix, 1);

    return S;
}

// Iterations for simplex method.
void Simplex(Simplex_Tableau *S) {

    // Pre-print the original Matrix.
    Matrix_print(S->Matrix);

    // Checking the Problem type.
    int i = 1;
    int count_minus = 0;
    for (; i <= S->Objective_Vector->n; i++) {
        double tmp = Dynamic_Array_get_Element(S->Objective_Vector, i);
        if (tmp < 1e-15 || tmp == 0) {
            count_minus += 1;
        }
    }
    if (count_minus == S->Objective_Vector->n) {
        printf("This Linear Programming MAYBE a <MIN> type\n");
        printf("Simplex Matrix will be reshaped.\n");
        for (i = 1; i <= S->Objective_Vector->n; i++) {
            if (Dynamic_Array_get_Element(S->Objective_Vector, i) != 0) {
                Matrix_pivot_Element_transInto_zero(S->Matrix, 1, i);
                Matrix_print(S->Matrix);
            }
        }
    }

```

```c
    int iter_deepth = 1;

    Dynamic_Array *object = Matrix_row_to_Vector(S->Matrix, 1, -1);
    int N_pivot_column = Dynamic_Array_find_Maximal(object);
    double Max = Dynamic_Array_get_Element(object, N_pivot_column);

    Dynamic_Array *pivot_column = Matrix_column_to_Vector(S->Matrix, N_pivot_column);
    Dynamic_Array *last_column = Matrix_column_to_Vector(S->Matrix, S->Matrix->n_column);

    Div_Dynamic_Array *tmp = Div_Dynamic_Array_init(last_column, pivot_column);
    int N_pivot_row = Div_Dynamic_Array_find_Minimal(tmp);

    Matrix_pivot_Element_Trans(S->Matrix, N_pivot_row, N_pivot_column);

    printf("Iter depth: %d\n", iter_deepth++);
    Matrix_print(S->Matrix);

    object = Matrix_row_to_Vector(S->Matrix, 1, -1);
    N_pivot_column = Dynamic_Array_find_Maximal(object);
    Max = Dynamic_Array_get_Element(object, N_pivot_column);

    while (Max > 0 && iter_deepth <=10000) {

        pivot_column = Matrix_column_to_Vector(S->Matrix, N_pivot_column);
        last_column = Matrix_column_to_Vector(S->Matrix, S->Matrix->n_column);

        tmp = Div_Dynamic_Array_init(last_column, pivot_column);
        N_pivot_row = Div_Dynamic_Array_find_Minimal(tmp);

        Matrix_pivot_Element_Trans(S->Matrix, N_pivot_row, N_pivot_column);
        printf("Iter depth: %d\n", iter_deepth);
        Matrix_print(S->Matrix);
        object = Matrix_row_to_Vector(S->Matrix, 1, -1);
        N_pivot_column = Dynamic_Array_find_Maximal(object);
        Max = Dynamic_Array_get_Element(object, N_pivot_column);

        iter_deepth += 1;
    }
}

// For finding a initial solution, dual simplex method is needed.
// The input may be a little bit complex.
void dual_Simplex(Simplex_Tableau *S, char *c2) {

    // First Phase
    S->Objective_Vector = Matrix_row_to_Vector(S->Matrix, 1, -1);
    Simplex(S);
    if (Matrix_get_Element(S->Matrix, 1, S->Matrix->n_column) > 1e-14) {
```

```
        printf("ANSWER of PHRASE ONE: %8.4f\n", Matrix_get_Element(S->Matrix, 1, S->Matrix->n_column));
        printf("No Feasible Solution.\n");
        return;
    }
    // else

    printf("First Phase completed.\n\n");
    Simplex_Tableau_re_init(S, c2);
    printf("NEW Objective Function is ");
    Dynamic_Array_print(S->Objective_Vector);
    printf("\n");
    Simplex(S);
}
```

程序代码 1

六、 运行结果

6.1 例 1

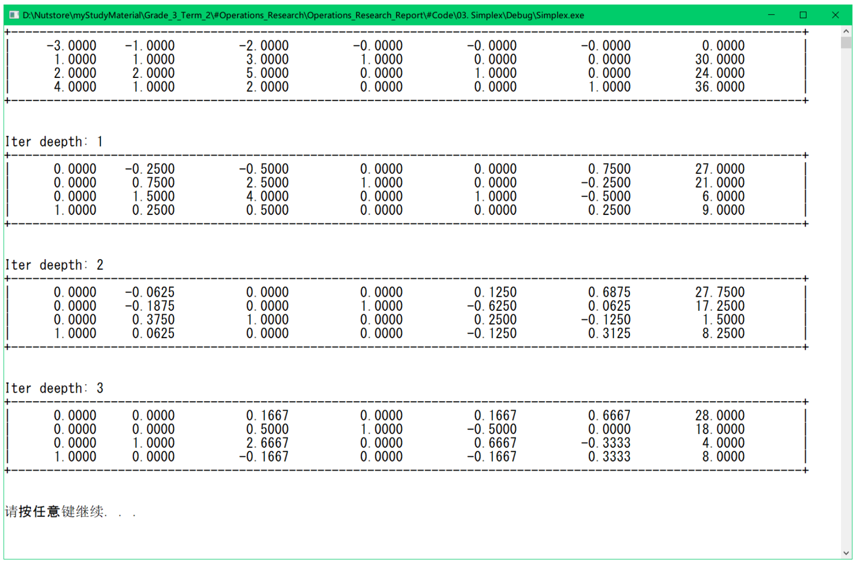
这个例子来自于算法导论[3]的 29.3 节

最大化 $$\max\ 3x_1 + x_2 + 2x_3$$

满足约束

$$\begin{aligned} x_1 + x_2 + 3x_3 &\le 30 \\ 2x_1 + 2x_2 + 5x_3 &\le 24 \\ 4x_1 + x_2 + 2x_3 &\le 36 \\ x_1,\ x_2,\ x_3 &\ge 0 \end{aligned}$$

6.1.1 运行结果



运行结果 1

6.1.2 代码分析

例子是算法导论[3]给出的一个例子，并没有采用单纯形两阶段法，因为都是小于等于的约束，所以初值很好取。

6.2 例 2

这个例子来自运筹学导论[1]两阶段的章节。

最小化
$$0.4x_1 + 0.5x_2$$
满足
$$\begin{aligned} 0.3x_1 + 0.1x_2 &\le 2.7 \\ 0.5x_1 + 0.5x_2 &\le 6 \\ 0.6x_1 + 0.4x_2 &\ge 6 \\ x_1,\ x_2 &\ge 0 \end{aligned}$$
因为存在非小于等于约束，所以先要把这些约束转化为等式约束

第一阶段问题
$$\min\ Z = \overline{x_4} + \overline{x_6}$$
s.t.
$$\begin{aligned} 0.3x_1 + 0.1x_2 + x_3 &= 2.7 \\ 0.5x_1 + 0.5x_2 + \overline{x_4} &= 6 \\ 0.6x_1 + 0.4x_2 - x_5 + \overline{x_6} &= 6 \end{aligned}$$
第二阶段问题
$$\min\ Z = 0.4x_1 + 0.5x_2$$
s.t.
$$\begin{aligned} 0.3x_1 + 0.1x_2 + x_3 &= 2.7 \\ 0.5x_1 + 0.5x_2 &= 6 \\ 0.6x_1 + 0.4x_2 - x_5 &= 6 \end{aligned}$$

6.2.1 运行结果

```
D:\Nutstore\myStudyMaterial\Grade_3_Term_2\#Operations_Research\Operations_Research_Report\#Code\03. Simplex\Debug\Simplex.exe
+------------------------------------------------------------------------------------------+
|     -0.0000   -0.0000      -0.0000      1.0000      -0.0000      1.0000      0.0000      |
|      0.3000    0.1000       1.0000      0.0000       0.0000      0.0000      2.7000      |
|      0.5000    0.5000       0.0000      1.0000       0.0000      0.0000      6.0000      |
|      0.6000    0.4000       0.0000      0.0000      -1.0000      1.0000      6.0000      |
+------------------------------------------------------------------------------------------+

This Linear Programming MAYBE a <MIN> type
Simplex Matrix will be reshaped.
+------------------------------------------------------------------------------------------+
|     -0.5000   -0.5000      -0.0000      0.0000      -0.0000      1.0000     -6.0000      |
|      0.3000    0.1000       1.0000      0.0000       0.0000      0.0000      2.7000      |
|      0.5000    0.5000       0.0000      1.0000       0.0000      0.0000      6.0000      |
|      0.6000    0.4000       0.0000      0.0000      -1.0000      1.0000      6.0000      |
+------------------------------------------------------------------------------------------+

+------------------------------------------------------------------------------------------+
|     -1.1000   -0.9000      -0.0000      0.0000       1.0000      0.0000    -12.0000      |
|      0.3000    0.1000       1.0000      0.0000       0.0000      0.0000      2.7000      |
|      0.5000    0.5000       0.0000      1.0000       0.0000      0.0000      6.0000      |
|      0.6000    0.4000       0.0000      0.0000      -1.0000      1.0000      6.0000      |
+------------------------------------------------------------------------------------------+

Iter depth: 1
+------------------------------------------------------------------------------------------+
|      0.0000   -0.5333       3.6667      0.0000       1.0000      0.0000     -2.1000      |
|      1.0000    0.3333       3.3333      0.0000       0.0000      0.0000      9.0000      |
|      0.0000    0.3333      -1.6667      1.0000       0.0000      0.0000      1.5000      |
|      0.0000    0.2000      -2.0000      0.0000      -1.0000      1.0000      0.6000      |
+------------------------------------------------------------------------------------------+

Iter depth: 2
+------------------------------------------------------------------------------------------+
|      0.0000    0.0000      -1.6667      0.0000      -1.6667      2.6667     -0.5000      |
|      1.0000    0.0000       6.6667      0.0000       1.6667     -1.6667      8.0000      |
|      0.0000    0.0000       1.6667      1.0000       1.6667     -1.6667      0.5000      |
|      0.0000    1.0000     -10.0000      0.0000      -5.0000      5.0000      3.0000      |
+------------------------------------------------------------------------------------------+

Iter depth: 3
+------------------------------------------------------------------------------------------+
|      0.0000    0.0000       0.0000      1.0000       0.0000      1.0000      0.0000      |
|      1.0000    0.0000       0.0000     -4.0000      -5.0000      5.0000      6.0000      |
|      0.0000    0.0000       1.0000      0.6000       1.0000     -1.0000      0.3000      |
|      0.0000    1.0000       0.0000      6.0000       5.0000     -5.0000      6.0000      |
+------------------------------------------------------------------------------------------+

First Phase completed.

NEW Objective Function is    -0.40       -0.50        0.00        0.00        0.00        0.00

+------------------------------------------------------------------------------------------+
|      0.4000    0.5000      -0.0000     -0.0000      -0.0000     -0.0000     -0.0000      |
|      1.0000    0.0000       0.0000      0.0000      -5.0000      0.0000      6.0000      |
|      0.0000    0.0000       1.0000      0.0000       1.0000      0.0000      0.3000      |
|      0.0000    1.0000       0.0000      0.0000       5.0000      0.0000      6.0000      |
+------------------------------------------------------------------------------------------+

This Linear Programming MAYBE a <MIN> type
Simplex Matrix will be reshaped.
+------------------------------------------------------------------------------------------+
|      0.0000    0.5000      -0.0000     -0.0000       2.0000     -0.0000     -2.4000      |
|      1.0000    0.0000       0.0000      0.0000      -5.0000      0.0000      6.0000      |
|      0.0000    0.0000       1.0000      0.0000       1.0000      0.0000      0.3000      |
|      0.0000    1.0000       0.0000      0.0000       5.0000      0.0000      6.0000      |
+------------------------------------------------------------------------------------------+

+------------------------------------------------------------------------------------------+
|      0.0000    0.0000      -0.0000     -0.0000      -0.5000     -0.0000     -5.4000      |
|      1.0000    0.0000       0.0000      0.0000      -5.0000      0.0000      6.0000      |
|      0.0000    0.0000       1.0000      0.0000       1.0000      0.0000      0.3000      |
|      0.0000    1.0000       0.0000      0.0000       5.0000      0.0000      6.0000      |
+------------------------------------------------------------------------------------------+

Iter depth: 1
+------------------------------------------------------------------------------------------+
|      0.0000    0.0000       0.5000      0.0000       0.0000      0.0000     -5.2500      |
|      1.0000    0.0000       5.0000      0.0000       0.0000      0.0000      7.5000      |
|      0.0000    0.0000       1.0000      0.0000       1.0000      0.0000      0.3000      |
|      0.0000    1.0000      -5.0000      0.0000       0.0000      0.0000      4.5000      |
+------------------------------------------------------------------------------------------+

请按任意键继续. . .
```

运行结果 2

6.2.2 代码分析

通过程序，得出了正确的答案。

七、 实验体会

这个单纯形算法并不是真正基于矩阵的算法。比较遗憾。然而这个程序有了循序渐进的单纯形算法的思维，基本思路借鉴了 Hiller 的参考书。

八、 参考文献

[1] HILLIER F S, LIEBERMAN G J. 运筹学导论 [M]. 9th ed. 北京: 清华大学出版社, 2010.

[2] 林锐. 高质量 C++/C 编程指南 [M]. 1.0 ed., 2001.

[3] CORMEN T H, LEISERSON C E, RIVEST R L, et al. 算法导论 [M]. 3rd ed. 北京: 机械工业出版社, 2013.