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： |  |

# 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。

# 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。

# 实验平台

Microsoft Windows 10 Pro Workstation 1803；

MathWorks MATLAB R2018a；

Microsoft Visual Studio 2017 Enterprise.

# 算法设计[[1]](#footnote-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，而且尽可能增加的值而不违反任何约束。变量称为基本变量，并且某个其他变量变成非基本变量。

|  |  |
| --- | --- |
| **Algorithm**  **Input**  **Output**  **Begin**  **Step 1**  **Step 2**  **Step 3**  **Step 4**  **Step 5**  **Step 6**  **Step 7**  **End** | **SIMPLEX**, Simplex Method for solving LP problems can start with .  (1) 系数矩阵，，是系数矩阵的第列;  (2) 价值向量;  (3) 常数向量;  It means to find , s.t. , and .  如果有最优解，输出最优解，如果没有，输出No Solution  ;  ;  ;  记的第一行为；  从中价值元素中的找寻最大的正数，命之为，记之在此行中的坐标为，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，，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，那么很显然并不行，因为它要求松弛变量与剩余变量的最终解都为，这是荒谬的，也就是说并不行，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，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，那说明只有这个不符合题意的解能单纯地满足约束方程），表现为的第一行有负数，，从中价值元素中的找寻最大的正数，命之为，记之在此行中的坐标为，设置本次迭代不在经过此步骤，否则会有死循环，GOTO **Step 6**;  利用条件除法作集合，，最大值不止一个就选其中一个，记在中的坐标为，GOTO **Step 7**  ;  for through to  if , then let  Go to **Step 5**； |

下面是调用了上面的单纯形算法的两阶段算法。

|  |  |
| --- | --- |
| **Algorithm**  **Input**  **Output**  **Begin**  **Step 1**  **Step 2**  **Step 3**  **End** | **DUAL-SIMPLEX**, 重点解决初值不能从开始的LP问题  (1) 系数矩阵，，是系数矩阵的第列;  (2) 价值向量;  (3) 价值向量;  (4) 常数向量;  如果有最优解，输出最优解，如果没有，输出No Solution  ，输出，此时如果，GOTO **Step 2**，若不为零，则输出无解，GOTO **End**;  重新初始化单纯形表以为第二阶段做准备：把中人工变量所在的行全部删除或者全部变为零，把第一行用代替；GOTO **Step 3**；  将传给**SIMPLEX**算法的**Step 4**，完成计算。 |

# 程序代码

## 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。

## 程序代码

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175 | /\*  \* 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  #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.  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**);**  **}**  **}**  **}**  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

# 运行结果

## 例1

这个例子来自于算法导论[3]的29.3节

最大化

满足约束

### 运行结果

图片包含 屏幕截图

已生成极高可信度的说明

运行结果 1

### 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。

## 例2

这个例子来自运筹学导论[1]两阶段的章节。

最小化

满足

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

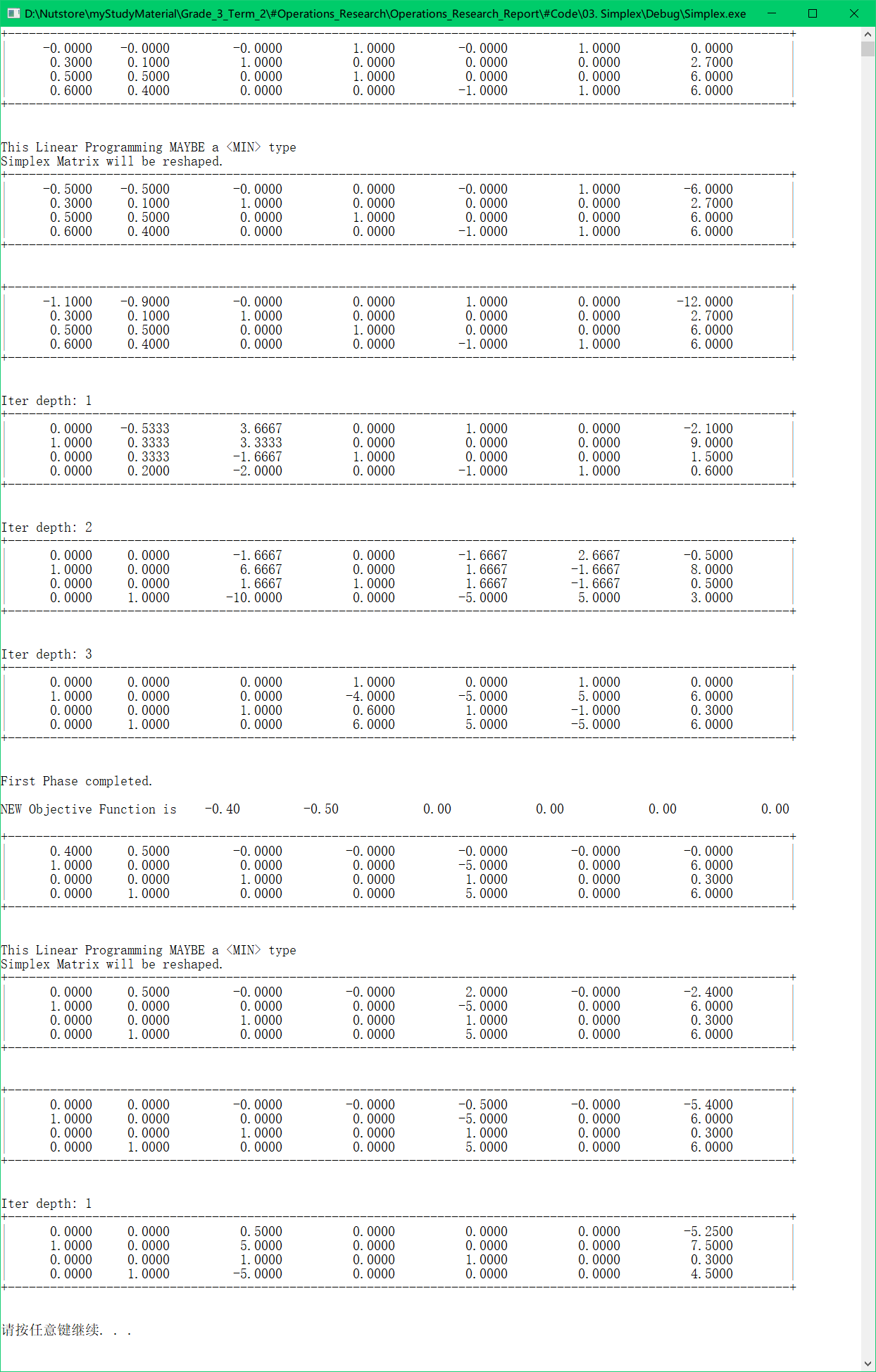
第一阶段问题

s.t.

第二阶段问题

s.t.

### 运行结果



运行结果 2

### 代码分析

通过程序，得出了正确的答案。

# 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。

# 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.

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。 [↑](#footnote-ref-1)