运筹学实验报告-1

姓名：李奕萱学号：PB22000161

2024 年 12 月 3 日

1 问题背景

线性规划（Linear Programming，简称 LP）作为运筹学的核心工具之一，旨在通过线性方程和不等式的约束条件，寻找一个最优的决策方案。线性规划问题的求解方法包括单纯形法、对偶单纯形法、内点法等，其中单纯形法是应用最广泛的算法之一。

然而，在实际问题中，线性规划往往伴随有特殊情况，如无可行解、无界解或冗余约束等，这些情况可能使得问题求解过程更加复杂。因此，在求解线性规划时，如何高效处理这些特殊情况并获得最优解，是运筹学中的一个重要课题。

本实验旨在通过单纯形法这种常见算法来求解线性规划问题，并通过实验验证这种算法的有效性、稳定性和计算效率。

2 算法介绍

2.1 前期准备

```
def pre_perform(A,b):
    check1,A1,b1=check_zero_rows(A,b)
    A2=check_zero_columns(A1)
    check2,A3,b3=check_linear_dependence(A2,b1)
    if (check1 and check2):
        return True,A3,b3
    else :
        return False,A3,b3
```

在进行线性规划求解之前，需要进行一些前期处理工作。首先，我们检查输入的矩阵是否存在全零行、全零列以及简单的线性相关行。如果有这些问题，需将对应的行或列删除，以简化问题的求解过程。

2.2 转化为标准形式

```python
def convert_to_standard_form( A, b,c, var_bounds=None):
    m, n = A.shape

    # Step 1: 将不等式约束 Ax <= b 转化为 Ax + s = b
    A_new = np.hstack([A, np.eye(m)])  # 引入松弛变量 s
    c_new = np.hstack([c, np.zeros(m)])  # 目标函数没有松弛变量
    b_new = b  # b 不变

    # Step 2: 处理变量边界，将无界变量拆分为差值形式
    if var_bounds is not None:
        for i in range(n):
            lb, ub = var_bounds[i]
            if lb == -np.inf and ub == np.inf:  # 无界变量
                # 将 x_i = x_i^+ - x_i^-
                A_new = np.hstack([A_new[:, :i], A_new[:, i:i+1], -A_new[:, i:i+1], A_new[:, i+1:]])
                c_new = np.hstack([c_new[:i], c_new[i:i+1], -c_new[i:i+1], c_new[i+1:]])

    return A_new, b_new,c_new
```

线性规划问题的一般形式为：

$$\text{Maximize } \mathbf{c}^T\mathbf{x}$$

subject to

$$\mathbf{A}\mathbf{x} \leq \mathbf{b}, \quad \mathbf{x} \geq 0$$

在实际应用中，约束条件往往是以不等式形式给出的，需要将其转化为等式形式。通过引入松弛变量，可以将不等式约束转化为等式约束，从而使问题能够采用标准的单纯形法进行求解。

此外，线性规划问题中可能存在无界变量，为了保证问题的求解稳定性，需要通过差分操作将无界变量转化为有界变量。

2.3 检查满秩

使用QR分解

- $A \in \mathbf{R}^{\{m\times n\}}$，记$B = A^T$
- 有如下分解：$B = QR$，其中Q$\in \mathbf{R}^{\{n\times m\}}$ 是正交矩阵，R的对角元的绝对值从大到小排列。
- 如果 R的对角元都非零，则A满秩。
- 反之，R有对角元为零，说明可以删掉对应的Q的列，得到B’。
- 最后得到满秩矩阵 $A' = (B')^T$
- 向量b删掉对应的行元素即可

```python
def check_full_rank(A,c):
    B=A.T
    Q,R=np.linalg.qr(B)
    dia=min(R.shape[0],R.shape[1])

    for i in range(dia):
        if R[i][i]!=0:
            continue
        else:
            R=np.delete(R,i,axis=0)
            R=np.delete(R,i,axis=1)
            Q=np.delete(Q,i,axis=1)
            c=np.delete(c,i,axis=0)

    A=(Q@R).T
    #A=np.round(A).astype(int)
    return A,c
```

对于线性规划问题的约束矩阵 **A**，我们需要检查其是否为满秩矩阵。如果矩阵存在冗余约束或线性相关的约束，这会影响问题求解的精度与稳定性。通过使用 QR 分解方法，可以检查矩阵是否为满秩矩阵，并进行必要的转换，以确保算法在执行时的有效性。

2.4 大 M 法获得初始可行基解

大 M 法考虑如下线性规划问题

$$\begin{aligned} \min \quad & c^\top x + M \cdot 1^\top y \\ \text{s.t.} \quad & Ax + y = b \\ & x \geq 0, y \geq 0. \end{aligned} \tag{4.7}$$

```python
def initialize_feasible_solution(A,M,b,c):

    m,n=A.shape

    A_new = np.hstack([A, np.eye(m)])
    c_new = np.hstack([c, M*np.ones(m)])

    x=np.hstack([np.zeros(n),b])

    return A_new,b,c_new,x
```

大 M 法是一种通过引入大数 M 以辅助求解初始可行解的方法。通过对人工变量的引入和对大数的设置，我们可以使得线性规划问题进入一个初始可行解。

2.5 单纯形法

单纯形算法 (SIMPLEX)

1. 第一步（初始化）：

   确定一个可行基划分 $A=(B,N)$，并计算 $x_B=\bar{b}:=B^{-1}b$.

2. 第二步（最优判定）：

   计算向量 $w=(B^\top)^{-1}c_B$，对所有的非基分量 $j\in N$ 求出 $z_j=w^\top A_j$.

   IF $\bar{c}_j=c_j-z_j\geq 0(j\in N)$，则当前的可行基解 $x=\begin{pmatrix} x_B \\ x_N \end{pmatrix}=\begin{pmatrix} \bar{b} \\ 0 \end{pmatrix}$ 已是最优解 [stop!]。

   ELSE 选取一个满足 $c_j-z_j<0$ 的 $j\in N$ 进入下一步。(**注：** 选取 j 的方式有多种。如：下标从

   小到大计算 $\bar{c}_j$，遇到小于 0 时直接选取。或者，计算出所有的情况，取 $\theta_j^*\bar{c}_j$ 中最小的那个。但是，后者在实际中并未提升求解效率。)

3. 第三步（转轴运算）：

   计算向量 $d=-B^{-1}A_j$.

   IF $d\geq 0$，则问题无有界解 [stop!!]。

   ELSE

   找出 r 使得 $\theta^*=\min_{\{i|d_{B_i}<0\}}(-\frac{x_{B_i}}{d_{B_i}})$，将基矩阵 B 的列向量 A_{B_r} 用 A_j 替换，得到新的基矩阵

   $$B=(A_{B_1},\cdots,A_{B_{r-1}},A_j,A_{B_{r+1}},\cdots,A_{B_m}).$$

   进而，记新基变量的值为

   $$x_{B_i}=x_{B_i}+\theta^* d_i(i=1,\cdots,m,i\neq r),x_j=\theta^*.$$

   令基变量的下标集合与非基变量的下标集合分别为

   $$B:=(B\cup\{j\})-\{B_r\},\quad N:=(N\cup\{B_r\})-\{j\}.$$

   回到第二步。

单纯形法是一种用于求解线性规划问题的常见算法，通过迭代计算在约束条件下不断优化目标函数，直至找到最优解。在本实验中，我们使用单纯形法进行求解，并在迭代过程中采用 **Bland’s Rule** 避免出现循环情况。

2.6 随机生成 Abc

```python
def generate(num_constraints,num_vars):
    # 随机生成目标函数系数
    c = np.random.randint(-10, 10, size=num_vars)

    # 随机生成约束矩阵 A 和右端项 b
    A = np.random.randint(-10, 10, size=(num_constraints, num
    b = np.random.randint(1, 20, size=num_constraints)
    return A,b,c
```

为了模拟实际应用中不同规模和结构的线性规划问题，我们使用 **random** 函数生成随机矩阵 A、b 和 c，并根据生成的随机矩阵求解线性规划问题。通过生成不同规模和类型的矩阵，实验能够验证算法在不同情况下的表现。

3 实验结果

3.1 有可行解

```
[[1 2 2]
 [2 1 2]
 [2 2 1]] [20 20 20]
 [-10 -12 -12]
solution: [4. 4. 4. 0. 0. 0. 0. 0. 0.]
optimal_value: -136.0
```

3.2 有冗余秩

```
[[1 2 3]
 [2 4 6]
 [1 1 1]] [ 6 12  3]
 [-1 -2 -3]
solution: [0. 3. 0. 0. 0. 0. 0.]
optimal_value: -6.0
```

3.3 无可行域

```
[[1 0 0]
 [0 1 0]
 [0 0 1]
 [1 0 1]] [2 2 2 2]
 [1 1 1]
LP has no feasible point
```

3.4 无界

```
[[ 1 -1]
 [-1  1]] [0 0]
 [-1  0]
unbounded
```

3.5 随机生成

```
[[  1  -3   5 -10]
 [  1  -7   4   0]
 [  8   2   6   3]] [14 14  6]
 [ 0 -8  5  1]
solution: [ 0.  3.  0.  0. 23. 35.  0.  0.  0.  0.]
optimal_value: -24.0
```

```
[[ 8  9 -7 -1]
 [-5  1  1 -3]
 [-3  5 -3 -4]] [ 3  7 14]
 [-9  5  2 -9]
unbounded
```

```
[[  5 -10   5  -8]
 [  4   9  -9 -10]
 [ -1   2   4   8]] [ 3  2 14]
 [-10  -6  -3  -5]
LP has no feasible point
```

下面是运行时间随着规模增大的拟合图像。

以 5 为步长，从 5 依次计算到 95，每种随机生成 20 组，仅统计有可行域的结果，对运行时间求平均值和方差得到 (纵轴单位为秒)。

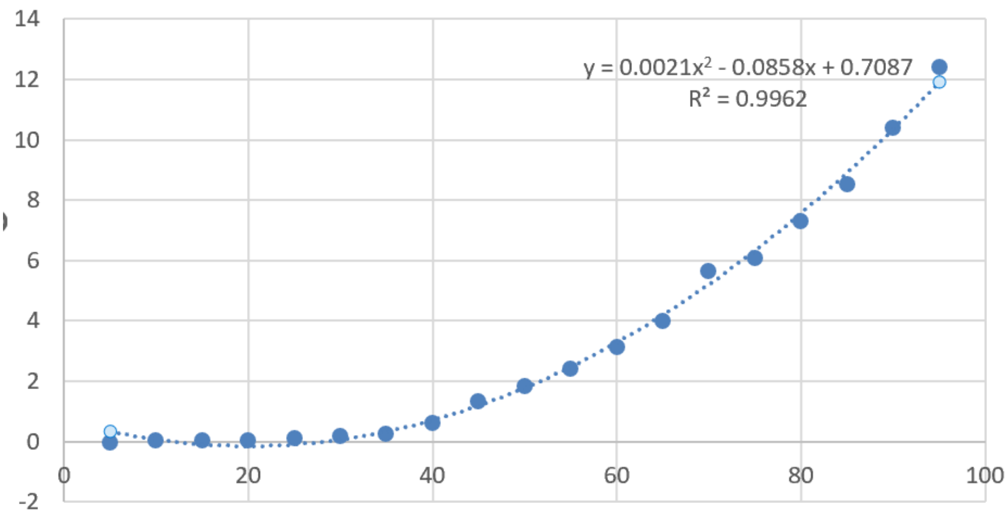

图 1: 时间均值-阶数

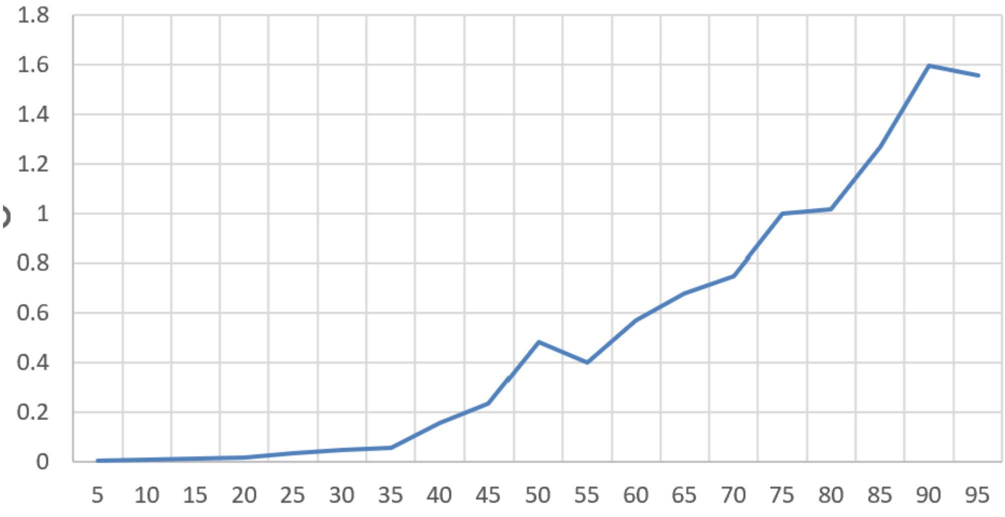

图 2: 时间方差-阶数

可以发现，随着阶数规模的增加，求解时间也在增加，大致为二阶线性。标准差也在增加，且波动明显大于平均时间。

4 总结

1. 在大 M 法中，初始可行基解的选择非常关键。如果选择不当，可能会导致算法运行时间过长，甚至进入死循环。我们采用 Bland’s Rule 来避免循环，这是一个非常有效的策略。
2. 矩阵规模与计算时间呈正相关（大致为二阶线性），随着矩阵规模的增大，计算时间显著增加，尤其是当约束条件较多时，计算时间增长更加明显。
3. 大 M 法和单纯形法是线性规划中常用的解法，对于小规模问题，两者的表现都较好。但随着规模的增加，求解时间较长时。
4. 特殊情况的处理：在处理无可行解、无界解等特殊情况时，实验系统能够及时反馈，并优化计算过程。