矩阵分析与应用大作业

一、程序总体介绍

本程序是中国科学院大学计算机学院矩阵分析与应用2021年秋天课程大作业。

共有7个功能：

- 矩阵的PLU分解
- 使用Gram-Schmidt正交化实现矩阵的QR分解
- 使用Householder正交约简实现矩阵的QR分解
- 使用Givens正交约简实现矩阵的QR分解
- 矩阵的URV分解
- 利用PLU分解计算矩阵行列式
- 利用QR分解求解线性方程组$Ax = b$

上述7个功能的函数分别位于`./methods`文件夹下。

`main.py`是程序主程序，也是入口。

`util.py`封装了一些共用函数，如打印矩阵、计算矩阵的秩、从文件读取矩阵。

二、使用方法

1. requirements

本程序只需要numpy包

使用命令`pip install numpy`安装即可。

2. 数据准备

将需要使用的矩阵数据写入`txt`文件，按照每一行的顺序写，每个元素直接用空格隔开(**注意每一行末尾不要留有空格**)，如下例子：

$$A = \begin{pmatrix} 1 & 2 & 4 & 17 \\ 3 & 6 & -12 & 3 \\ 2 & 3 & -3 & 2 \\ 0 & 2 & -2 & 6 \end{pmatrix}$$

矩阵A在文件中写为：

```
1 2 4 17
3 6 -12 3
2 3 -3 2
0 2 -2 6
```

3. 程序运行方法

建议在终端运行：

在终端下将当前目录切换到`./MatrixFinalProject`，然后使用指令`python main.py`启动程序即可。

根据提示输入$0-6$的序号选择对应的功能，然后根据提示输入数据的文件名后回车即可运行。

```
请选择需要执行的方法编号:
0 : PLU分解(输入必须是可逆的方阵, 输出分别是P,L,U矩阵)
1 : Gram_Schmidt正交化求QR分解(输入必须是列向量无关的矩阵,输出分别是Q,R矩阵)
2 : Householder约简求QR分解(输入为m*n的矩阵,输出分别是Q,R矩阵)
3 : Givens约简求QR分解(输入为m*n的矩阵,输出分别是Q,R矩阵)
4 : URV分解(输入为m*n的矩阵,输出分别是U,R,V矩阵)
5 : 求矩阵的行列式(输入必须为方阵,输出是行列式值)
6 : 求线性方程组Ax=b(输入必须为m*n的系数矩阵A和m*1的向量b,输出是x,若无解则输出最小二乘解)
6
请输入需要读取数据的文件名, 直接回车默认为'data.txt':

请输入需要读取Ax=b的b数据的文件名, 直接回车默认为'b.txt':

系数矩阵:
    1.00      2.00      4.00     17.00
    3.00      6.00    -12.00      3.00
    2.00      3.00     -3.00      2.00
    0.00      2.00     -2.00      6.00
b:
   17.00      3.00      3.00      4.00
x:
    2.00     -1.00     -0.00      1.00
```

三、程序功能详细介绍

1. 矩阵的PLU分解

对一个$n\times n$大小的可逆方阵进行PLU分解。

`Input: A (n*n的待分解可逆方阵)`

`Output: P(n*n的置换矩阵), L(n*n的下三角矩阵), U(n*n的上三角矩阵)`

函数所在位置`./methods/LU.py`，函数名为`PLU_Factorization`

例子：

```
请选择需要执行的方法编号:
0 : PLU分解(输入必须是可逆的方阵, 输出分别是P,L,U矩阵)
1 : Gram_Schmidt正交化求QR分解(输入必须是列向量无关的矩阵,输出分别是Q,R矩阵)
2 : Householder约简求QR分解(输入为m*n的矩阵,输出分别是Q,R矩阵)
3 : Givens约简求QR分解(输入为m*n的矩阵,输出分别是Q,R矩阵)
4 : URV分解(输入为m*n的矩阵,输出分别是U,R,V矩阵)
5 : 求矩阵的行列式(输入必须为方阵,输出是行列式值)
6 : 求线性方程组Ax=b(输入必须为m*n的系数矩阵A和m*1的向量b,输出是x,若无解则输出最小二乘解)
0
请输入需要读取数据的文件名, 直接回车默认为'data.txt':

原矩阵:
    1.00     2.00     4.00    17.00
    3.00     6.00   -12.00     3.00
    2.00     3.00    -3.00     2.00
    0.00     2.00    -2.00     6.00
交换矩阵P:
    0.00     1.00     0.00     0.00
    0.00     0.00     0.00     1.00
    1.00     0.00     0.00     0.00
    0.00     0.00     1.00     0.00
L矩阵:
    1.00     0.00     0.00     0.00
    0.00     1.00     0.00     0.00
    0.33     0.00     1.00     0.00
    0.67    -0.50     0.50     1.00
U矩阵:
    3.00     6.00   -12.00     3.00
    0.00     2.00    -2.00     6.00
    0.00     0.00     8.00    16.00
    0.00     0.00     0.00    -5.00
验证正确性: (A=P^T*L*U)
    1.00     2.00     4.00    17.00
    3.00     6.00   -12.00     3.00
    2.00     3.00    -3.00     2.00
    0.00     2.00    -2.00     6.00
```

详细代码及注释:

算法过程就是使用部分主元法的高斯消元法求LU分解，同时对交换矩阵P进行同样的交换操作，用来保存最终的顺序。

```python
def PLU_Factorization(A):
    # 输入一个 n*n 的方阵，使用numpy格式
    # 输出PLU分解后的三个矩阵，分别是 P， L， U
    A = A.copy()
    if A is None or (A.shape[0] != A.shape[1]):  # 判断是否为方阵
        return 0, 0, 0     # 不满足条件，直接返回，返回数字为了后续判断
    n = A.shape[0]    # 矩阵的维度n
    if calculate_rank(A) != n:  # 判断矩阵是否可逆
        return 1, 1, 1     # 不满足条件，直接返回，返回数字为了后续判断
    P = np.eye(n)     # 交换矩阵P
    for j in range(n):
        index = np.argmax(np.abs(A[j:n, j]))    # 找到主元最大的行
        A[j, :], A[j+index, :] = A[j+index, :].copy(), A[j, :].copy()     # 交换行
        P[j, :], P[j+index, :] = P[j+index, :].copy(), P[j, :].copy()     # 同时交换矩阵P
        for i in range(j+1, n):     # 对主元下面的行进行消元操作
            a = A[i, j] / A[j, j]         # 计算消元系数
            A[i, j:] = A[i, j:] - a * A[j, j:]       # 消元
            A[i, j] = a      # 记录消元系数
    L = np.eye(n) + np.tril(A, -1)   # 最后L为A的下三角（去除主对角线）加上一个单位阵
    U = np.triu(A)   # U为A的上三角及其主对角线
    return P, L, U
```

2.使用Gram-Schmidt正交化实现矩阵的QR分解

使用Gram-Schmidt正交化的方法对任意大小$m \times n$的矩阵做QR分解，但**必须保证矩阵的列向量之间线性无关。**

`Input: A (m*n的待分解矩阵)`

`Output: Q(m*n的标准正交矩阵), R(n*n的上三角矩阵)`

函数所在位置`./methods/Gram_Schmidt.py`，函数名为`gram_schmidt`

例子:

```
请选择需要执行的方法编号:
0 : PLU分解(输入必须是可逆的方阵, 输出分别是P,L,U矩阵)
1 : Gram_Schmidt正交化求QR分解(输入必须是列向量无关的矩阵,输出分别是Q,R矩阵)
2 : Householder约简求QR分解(输入为m*n的矩阵,输出分别是Q,R矩阵)
3 : Givens约简求QR分解(输入为m*n的矩阵,输出分别是Q,R矩阵)
4 : URV分解(输入为m*n的矩阵,输出分别是U,R,V矩阵)
5 : 求矩阵的行列式(输入必须为方阵,输出是行列式值)
6 : 求线性方程组Ax=b(输入必须为m*n的系数矩阵A和m*1的向量b,输出是x,若无解则输出最小二乘解)
1
请输入需要读取数据的文件名, 直接回车默认为'data.txt':

原矩阵:
    1.00      2.00      4.00     17.00
    3.00      6.00    -12.00      3.00
    2.00      3.00     -3.00      2.00
    0.00      2.00     -2.00      6.00
Q矩阵:
    0.27      0.07      0.88      0.38
    0.80      0.20     -0.42      0.38
    0.53     -0.33      0.19     -0.76
    0.00      0.92      0.09     -0.38
R矩阵:
    3.74      6.95    -10.16      8.02
    0.00      2.17     -2.96      6.58
    0.00      0.00      7.82     14.70
    0.00      0.00      0.00      3.78
验证正确性(Q*R):
    1.00      2.00      4.00     17.00
    3.00      6.00    -12.00      3.00
    2.00      3.00     -3.00      2.00
    0.00      2.00     -2.00      6.00
```

详细代码及注释:

算法的过程是逐列进行施密特正交化，每次减去此行对前面已经计算出的正交向量的投影，然后归一化。

```python
def gram_schmidt(A):
    # 输入一个 n*m 的矩阵，必须列向量无关，使用numpy格式
    # 输出二个矩阵，分别是 Q(n*m), R(m*m)
    n, m = A.shape  #  矩阵的形状
    # 先判断是否列向量无关:
    # 首先求矩阵A的秩
    r = calculate_rank(A)
    if r != m:
        print("输入矩阵不是列向量无关的!")
        return 0, 0

    Q = np.zeros((n, m))    # 建立一个空的Q矩阵, 形状为n*m
    R = np.zeros((m, m))    # 建立一个空的R矩阵, 形状为m*m
    for i in range(m):      # 循环处理A的每一列
        q = np.array(A[:,i]).squeeze()   # 提出A的第i列
```

```python
        for j in range(0, i):          # 循环对这一列减去他对前面已经计算出的正交向量的投影
            t = (Q[:,j]*q).sum()      # 求内积
            R[j, i] = t       # 内积赋到对应的R矩阵上
            q -= t * Q[:,j]     #  减去前面已经计算出的正交向量的投影
        norm = np.sqrt(np.power(q, 2).sum())     # 计算正交向量的模
        if norm != 0:      # 若模长不为0，则归一化
            q /= norm
        R[i, i] = norm     # 模长赋到R矩阵对应的位置上
        Q[:,i] = q.ravel()    # 将计算好的正交向量赋到Q矩阵里
    return Q, R
```

3. 使用Householder正交约简实现矩阵的QR分解

使用Householder正交约简的方法对任意大小$m \times n$的矩阵做QR分解。

`Input: A (m*n的待分解矩阵)`

`Output: Q(m*m的正交矩阵), R(m*n的上三角矩阵)`

函数所在位置`./methods/Householder_Reduction.py`，函数名为`house_holder`

例子:

```
请选择需要执行的方法编号:
0 : PLU分解(输入必须是可逆的方阵，输出分别是P,L,U矩阵)
1 : Gram_Schmidt正交化求QR分解(输入必须是列向量无关的矩阵,输出分别是Q,R矩阵)
2 : Householder约简求QR分解(输入为m*n的矩阵,输出分别是Q,R矩阵)
3 : Givens约简求QR分解(输入为m*n的矩阵,输出分别是Q,R矩阵)
4 : URV分解(输入为m*n的矩阵,输出分别是U,R,V矩阵)
5 : 求矩阵的行列式(输入必须为方阵,输出是行列式值)
6 : 求线性方程组Ax=b(输入必须为m*n的系数矩阵A和m*1的向量b,输出是x,若无解则输出最小二乘解)
2
请输入需要读取数据的文件名，直接回车默认为'data.txt':

原矩阵:
    1.00      2.00      4.00     17.00
    3.00      6.00    -12.00      3.00
    2.00      3.00     -3.00      2.00
    0.00      2.00     -2.00      6.00
Q矩阵:
    0.27      0.07      0.88     -0.38
    0.80      0.20     -0.42     -0.38
    0.53     -0.33      0.19      0.76
    0.00      0.92      0.09      0.38
R矩阵:
    3.74      6.95    -10.16      8.02
    0.00      2.17     -2.96      6.58
    0.00      0.00      7.82     14.70
   -0.00     -0.00      0.00     -3.78
验证正确性(Q*R):
    1.00      2.00      4.00     17.00
    3.00      6.00    -12.00      3.00
    2.00      3.00     -3.00      2.00
    0.00      2.00     -2.00      6.00
```

详细代码及注释:

逐列构造反射矩阵，然后依次与原矩阵相乘，但是对$m \times n$的矩阵操作时，第一次对原矩阵进行，第二次就选择去掉第一行第一列，对$m-1 \times n-1$的子矩阵进行操作，以此类推。子矩阵的反射矩阵用单位阵去填充，使其大小一样。

```python
def house_holder(A):
    # 输入一个n*m 的矩阵，使用numpy格式
    # 输出house_holder约减产生的Q(n*n),R(n*m)矩阵
```

```python
    n, m = A.shape  #  矩阵的形状
    Q = np.identity(n)  # Q矩阵初始化为单位阵
    R = A.copy()  # R矩阵初始化为原始矩阵A
    for i in range(min(n-1, m)):   #  构造的反射矩阵的个数是 min(行数-1,  列数)
        u = R[i:, i].copy()
        u[0] -= np.sqrt(np.power(u, 2).sum())   #  u = x - ||x||e_1
        r = np.identity(n)    # 将反射矩阵初始化为和R一样大的单位阵
        # 这里巧妙的将反射矩阵初始化为n*n, 然后只对要进行约减的地方变换为反射矩阵, 这样可以直接构造出n*n大小的反射矩阵
        norm = u.T @ u
        if norm != 0:
            r[i:, i:] -= 2 * (u @ u.T) / norm  # r = I - 2dd^T / d^Td
        # else:
            # r[i:, i:] -= 2 * (u @ u.T)
        R = r @ R    # 对原矩阵进行约减, 最终会得到矩阵R
        Q = Q @ r    # 因为Q = (r_3 r_2 r_1)^T = r_1 r_2 r_3 , 所以只需要将按顺序将生成的反射矩阵相乘即可得到最终的Q
    return Q, R
```

4.使用Givens正交约简实现矩阵的QR分解

使用Givens正交约简的方法对任意大小$m \times n$的矩阵做QR分解。

`Input: A (m*n的待分解矩阵)`

`Output: Q(m*m的正交矩阵), R(m*n的上三角矩阵)`

函数所在位置`./methods/Givens_Reduction.py`, 函数名为`givens`

例子:

```
请选择需要执行的方法编号:
0 : PLU分解(输入必须是可逆的方阵, 输出分别是P,L,U矩阵)
1 : Gram_Schmidt正交化求QR分解(输入必须是列向量无关的矩阵,输出分别是Q,R矩阵)
2 : Householder约简求QR分解(输入为m*n的矩阵,输出分别是Q,R矩阵)
3 : Givens约简求QR分解(输入为m*n的矩阵,输出分别是Q,R矩阵)
4 : URV分解(输入为m*n的矩阵,输出分别是U,R,V矩阵)
5 : 求矩阵的行列式(输入必须为方阵,输出是行列式值)
6 : 求线性方程组Ax=b(输入必须为m*n的系数矩阵A和m*1的向量b,输出是x,若无解则输出最小二乘解)
3
请输入需要读取数据的文件名, 直接回车默认为'data.txt':

原矩阵:
    1.00      2.00      4.00     17.00
    3.00      6.00    -12.00      3.00
    2.00      3.00     -3.00      2.00
    0.00      2.00     -2.00      6.00
Q矩阵:
    0.27      0.07      0.88      0.38
    0.80      0.20     -0.42      0.38
    0.53     -0.33      0.19     -0.76
    0.00      0.92      0.09     -0.38
R矩阵:
    3.74      6.95    -10.16      8.02
    0.00      2.17     -2.96      6.58
   -0.00     -0.00      7.82     14.70
   -0.00     -0.00     -0.00      3.78
验证正确性(Q*R):
    1.00      2.00      4.00     17.00
    3.00      6.00    -12.00      3.00
    2.00      3.00     -3.00      2.00
    0.00      2.00     -2.00      6.00
```

详细代码及注释:

构造旋转矩阵，使得矩阵每个主元(i,i)下面的元素都变为0，对一个$m\times n$的矩阵来说，第一列构造$m-1$个旋转矩阵，第二列构造$m-2$个旋转矩阵，以此类推，最终得到Q,R分解。

```
def givens(A):
    # 输入一个n*m 的矩阵，使用numpy格式
    # 输出Givens约减产生的Q(n*n),R(n*m)矩阵
    n, m = A.shape  #  矩阵的形状
    Q = np.identity(n)  # Q矩阵初始化为单位阵
    R = A.copy()  # R矩阵初始化为原始矩阵A
    for i in range(m):   #  对每一列做化简
        for j in range(i+1, n):     # 将主对角元素下面的每一个元素化为0
            if R[j, i] == 0:  # 若已经是0则跳过
                continue
            norm = np.sqrt(R[i, i] * R[i, i] + R[j, i] * R[j, i])  # 构造旋转矩阵，分别结算C和S
            c = R[i, i] / norm
            s = R[j, i] / norm
            P_ij = np.identity(n)    #  初始化旋转矩阵
            P_ij[i, i], P_ij[i, j], P_ij[j, i], P_ij[j, j] = c, s, -s, c  # 填入C和S
            R = P_ij @ R    # 对原矩阵进行约简
            Q = P_ij @ Q    # Q = (P23 P13 P12)^T
    Q = Q.T
    return Q, R
```

5.矩阵的URV分解

对任意大小$m\times n$的矩阵A做URV分解，得到矩阵U,R,V：

U是$m\times m$的正交矩阵，V是$n\times n$的正交矩阵，R是$\begin{pmatrix} C_{r\times r} & 0 \\ 0 & 0 \end{pmatrix}$，其中$r$是矩阵A的秩。

`Input: A (m*n的待分解矩阵)`

`Output: U(m*m的正交矩阵), V(n*n的正交矩阵), R(左上角是一个r*r的子矩阵，其余地方为0)`

函数所在位置`./methods/URV.py`，函数名为`URV`

例子：

```
请选择需要执行的方法编号:
0 : PLU分解(输入必须是可逆的方阵, 输出分别是P,L,U矩阵)
1 : Gram_Schmidt正交化求QR分解(输入必须是列向量无关的矩阵,输出分别是Q,R矩阵)
2 : Householder约简求QR分解(输入为m*n的矩阵,输出分别是Q,R矩阵)
3 : Givens约简求QR分解(输入为m*n的矩阵,输出分别是Q,R矩阵)
4 : URV分解(输入为m*n的矩阵,输出分别是U,R,V矩阵)
5 : 求矩阵的行列式(输入必须为方阵,输出是行列式值)
6 : 求线性方程组Ax=b(输入必须为m*n的系数矩阵A和m*1的向量b,输出是x,若无解则输出最小二乘解)
4
请输入需要读取数据的文件名, 直接回车默认为'data.txt':
urv.txt
矩阵的秩为:  2
原矩阵:
  -4.00    -2.00     4.00     2.00
   2.00    -2.00    -2.00    -1.00
  -4.00     1.00     4.00     2.00
U矩阵:
  -0.67    -0.67    -0.33
   0.33    -0.67     0.67
  -0.67     0.33     0.67
R矩阵:
   9.00    -0.00     0.00     0.00
  -0.00     3.00     0.00     0.00
   0.00     0.00     0.00     0.00
V矩阵:
   0.67     0.00     0.67     0.33
   0.00     1.00    -0.00    -0.00
  -0.67     0.00     0.73    -0.13
  -0.33     0.00    -0.13     0.93
验证正确性(U*R*V^T):
  -4.00    -2.00     4.00     2.00
   2.00    -2.00    -2.00    -1.00
  -4.00     1.00     4.00     2.00
```

详细代码及注释:

U的前r列是$R(A)$的标准正交基

U的后$m-r$列是$N(A^T)$的标准正交基

V的前r列是$R(A^T)$的标准正交基

V的后$n-r$列是$N(A)$的标准正交基

但由于零空间的标准正交基不容易计算，因此可以利用Householder方法计算：

首先使用Householder约简得到一个正交矩阵P使得$PA=\begin{pmatrix}B\\0\end{pmatrix}$，其中$B$是一个$r\times n$的矩阵，然后再对$B^T$使用Householder约简得到一个矩阵$Q$使得$QB^T=\begin{pmatrix}T\\0\end{pmatrix}$，其中$T$是一个$r\times r$的矩阵。因此，$B=\begin{pmatrix}T^T & 0\end{pmatrix}Q$，进而得到$\begin{pmatrix}B\\0\end{pmatrix}=\begin{pmatrix}T^T & 0\\0 & 0\end{pmatrix}Q$，最终可得到$A=P^T\begin{pmatrix}T^T & 0\\0 & 0\end{pmatrix}Q$。便可实现$URV^T$分解，其中$U=P^T,V=Q^T,R=\begin{pmatrix}T^T & 0\\0 & 0\end{pmatrix}$。

```
def URV(A):
    # 输入一个m*n 的矩阵，使用numpy格式
    # 输出URV分解产生的R(m*m),R(m*n),V(n*n)矩阵
    m, n = A.shape  #  矩阵的形状
    # 首先求矩阵A的秩
    r = calculate_rank(A)
    print("矩阵的秩为：", r)

    A = A.copy()
    Q1, R1 = house_holder(A)   # 使用householder约简得到 PA = (B, 0)^T
    P = Q1.T
    B = R1[:r, :]    # B是r*n的矩阵
```

```python
    Q2, R2 = house_holder(B.T)     # 然后对B^T做householder约简，得到QB^T = (T, 0)^T
    Q = Q2.T
    T = R2[:r, :]     # T是 r*r的矩阵
    # 最终可得到 A = P^T (T^T 0 \\ 0 0) Q
    # 对应A = U R V^T  可得   U = P.T , R =  (T^T 0 \\ 0 0) , V = Q.T
    U = P.T
    R = np.zeros((m, n))
    R[:r, :r] = T.T
    V = Q.T
    return U, R, V
```

6.计算矩阵行列式

对$n \times n$的矩阵A计算行列式值：

`Input: A (n*n的矩阵)`

`Output: det(A)`

函数所在位置`./methods/Determinant.py`，函数名为`det`

例子：

```
请选择需要执行的方法编号：
0 ： PLU分解(输入必须是可逆的方阵，输出分别是P,L,U矩阵)
1 ： Gram_Schmidt正交化求QR分解(输入必须是列向量无关的矩阵,输出分别是Q,R矩阵)
2 ： Householder约简求QR分解(输入为m*n的矩阵,输出分别是Q,R矩阵)
3 ： Givens约简求QR分解(输入为m*n的矩阵,输出分别是Q,R矩阵)
4 ： URV分解(输入为m*n的矩阵,输出分别是U,R,V矩阵)
5 ： 求矩阵的行列式(输入必须为方阵,输出是行列式值)
6 ： 求线性方程组Ax=b(输入必须为m*n的系数矩阵A和m*1的向量b,输出是x,若无解则输出最小二乘解)
5
请输入需要读取数据的文件名，直接回车默认为'data.txt'：

原矩阵：
    1.00      2.00      4.00     17.00
    3.00      6.00    -12.00      3.00
    2.00      3.00     -3.00      2.00
    0.00      2.00     -2.00      6.00
矩阵A的行列式等于：  240.0
```

详细代码及注释:

算法思路是先对原矩阵A进行PLU分解，由于P的行列式是正负一，L的是1，U的是主对角线元素乘积，因此在PLU分解的时候记录一下交换次数，交换奇数次是-1，交换偶数次是1，所以A的行列式值就是R的对角线元素乘符号。

```python
def det(A):
    # 求矩阵A的行列式，使用PLU分解求，P的det根据交换次数来算，交换次数为偶数则是1，为奇数则是-1
    # L的det是1， U的det是主对角线元素的乘积
    # PA = LU, A = P^{-1}LU = PLU
    # det(A) = det(PLU) = det(P)det(U) = 正负1 * U的对角线元素乘积
    A = A.copy()
    if A is None or (A.shape[0] != A.shape[1]):  # 判断是否为方阵
        return "error"
    n = A.shape[0]    # 矩阵的维度n
    if calculate_rank(A) != n:  # 判断矩阵是否可逆
        return 0
    P = 0  # 交换次数
```

```python
    for j in range(n):
        index = np.argmax(np.abs(A[j:n, j]))   # 找到主元最大的行
        if index != 0:
            A[j, :], A[j+index, :] = A[j+index, :].copy(), A[j, :].copy()   # 交换行
            P += 1   # 记录交换次数
        for i in range(j+1, n):    # 对主元下面的行进行消元操作
            a = A[i, j] / A[j, j]       # 计算消元系数
            A[i, j:] = A[i, j:] - a * A[j, j:]      # 消元
    # P是交换次数，交换奇数次是-1，交换偶数次是1，所以A的行列式值就是R的对角线元素乘符号
    Det = np.prod(np.diag(A)) * ((-1) ** P)
    return Det
```

7.求解线性方程组$Ax = b$

:

求解线性方程组$Ax=b$，其中系数矩阵A是$m\times n$的，b是$m\times 1$的，需要求解n个未知数x

`Input: A (m*n的系数矩阵),  b(m*1的向量)`

`Output: 有解则输出解，无解则输出最小二乘解 x(n*1的向量)`

函数所在位置`./methods/Linear_Equations.py`，函数名为`linear_equations`

例子:

```
请选择需要执行的方法编号:
0 : PLU分解(输入必须是可逆的方阵,输出分别是P,L,U矩阵)
1 : Gram_Schmidt正交化求QR分解(输入必须是列向量无关的矩阵,输出分别是Q,R矩阵)
2 : Householder约简求QR分解(输入为m*n的矩阵,输出分别是Q,R矩阵)
3 : Givens约简求QR分解(输入为m*n的矩阵,输出分别是Q,R矩阵)
4 : URV分解(输入为m*n的矩阵,输出分别是U,R,V矩阵)
5 : 求矩阵的行列式(输入必须为方阵,输出是行列式值)
6 : 求线性方程组Ax=b(输入必须为m*n的系数矩阵A和m*1的向量b,输出是x,若无解则输出最小二乘解)
6
请输入需要读取数据的文件名，直接回车默认为'data.txt':

请输入需要读取Ax=b的b数据的文件名，直接回车默认为'b.txt':

系数矩阵:
    1.00      2.00      4.00     17.00
    3.00      6.00    -12.00      3.00
    2.00      3.00     -3.00      2.00
    0.00      2.00     -2.00      6.00
b:
   17.00      3.00      3.00      4.00
x:
    2.00     -1.00     -0.00      1.00
```

详细代码及注释:

通过QR分解求线性方程组，即$QRx=b\rightarrow Rx=Q^Tb$，其中R是上三角矩阵，因此可以自下而上逐个计算x。

由于使用Householder约简求出的Q，R，若Q中有线性相关列时，R中会出现全0行，因此先将R中的全零行删除，再将Q中对应的列删除，然后再求解$Rx=Q^Tb$。

```python
def linear_equations(A, b):
    # 使用QR分解求线性方程组Ax=b
    # 输入是一个 m*n 的系数矩阵A，以及m*1的向量b
    # 输出是 n*1的向量，表示n个未知数x的值
    # 先使用 householder约简求出Q，R
```

```python
    Q, R = house_holder(A)
    # R可能存在全0行，将其去除掉，同时在Q中去掉对应的线性相关列
    mask = (np.abs(R) <= 1e-8)     # 判断每个元素是否是0
    p = np.where(mask.all(axis=1))[0]    # 找到全0行的index
    R = np.delete(R, p, 0)     # R矩阵删除全0的行
    Q = np.delete(Q, p, 1)     # Q矩阵删除全0的列
    # Ax = b  =>  QRx=b  =>  Rx = Q^T b
    b = Q.T @ b
    m, n = R.shape

    x = np.zeros(n)
    # 自下向上逐个求x_n, x_{n-1}, ... , x_2, x_1
    for i in range(m-1, -1, -1):
        x[i] = b[i]
        for j in range(i+1, n):
            x[i] -= x[j] * R[i, j]
        x[i] /= R[i, i]
    return x
```