作业七：矩阵分解与方程求解报告

学号： 202128013229021

姓名： 刘炼

程序说明

运行环境

python 3.6 及以上

依赖包

numpy

argparse

程序介绍

一个综合程序，根据选择参数的不同，实现不同的矩阵分解；在此基础上，实现Ax=b方程组的求解，以及计算A的行列式；

下面给出不同矩阵分解的计算以及利用此实现的方程求解

LU 分解

分解条件和思路

分解的基本条件如下：

1. 是一个square matrix ($n \times n$)
2. 在分解过程中，任意主元都不能为0

具体计算思路为： 将输入矩阵A利用初等变换的方式进行部分主元的高斯消去法，在变换的同时将执行列取代运算的乘数记录。最终根据运算，得到分解后的结果可以表示为PA=LU。在具体计算过程中，首先要判断输入矩阵是否是方阵，并且在计算过程中，要判定是否有为0 的主元，如果存在，则报错。

分解后矩阵的性质

分解得到的矩阵P, L, U 分别具有如下的性质：

1. L 为下三角矩阵，且U为上三角矩阵，P为行变换矩阵

2. 对于$i=1,\ldots,n, L_{ii}=1$且$U_{ii}\neq 0$
3. $PA=LU$，其中A为输入矩阵。

方程求解

对于原来的方程$Ax=b$， 其中$PA=LU$，所以原来的方程可以变为$LUx=Pb$，进一步推导为:

$$y=Ux=L^{-1}Pb$$
$$x=U^{-1}y$$

由于L 和 U 分别为下三角矩阵和上三角矩阵，对于y进行计算不需要再进行高斯消去操作，可以很简便进行计算。

实例

```
[[11. 10.  4. 18. 18.]
 [ 5.  2.  8. 16. 14.]
 [16. 13.  0.  2. 14.]
 [11. 15. 19.  8.  2.]
 [ 9. 11.  8. 17.  8.]]
----------the LU FACTORIZATION RESULTS----------
P
[[0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0.]]
L
[[ 1.     0.     0.     0.     0.   ]
 [ 0.688  1.     0.     0.     0.   ]
 [ 0.312 -0.34   1.     0.     0.   ]
 [ 0.562  0.608 -0.246  1.     0.   ]
 [ 0.688  0.175  0.046  0.905  1.   ]]
U
[[16.    13.     0.     2.    14.   ]
 [ 0.     6.062 19.     6.625 -7.625]
 [ 0.     0.    14.464 17.629  7.031]
 [ 0.     0.     0.    16.18   6.492]
 [ 0.     0.     0.     0.     3.509]]
original results
[[16. 13.  0.  2. 14.]
 [11. 15. 19.  8.  2.]
 [ 5.  2.  8. 16. 14.]
 [ 9. 11.  8. 17.  8.]
 [11. 10.  4. 18. 18.]]
----------b----------
[ 5. 14. 15. 17. 14.]
----------x: equation results----------
[ 8.307 -5.942  0.465  1.521 -3.122]
```

程序代码

```
def change_row(matrix,i,j):
    '''
    :param matrix: 输入矩阵
    :param i, j:需交换的行
    :return:  交换后的矩阵
    '''
```

```python
        matrix[[i,j], :] =  matrix[[j,i], :]
    return matrix

def PLUFactorization(matrix):
    '''
    :param matrix: 输入矩阵
    :param P L U: 分解得到的输出矩阵
    '''
    U = np.copy(matrix)
    row_len=U.shape[0]
    col_len=U.shape[1]

    assert (row_len == col_len), "PLU Factorization needs
a square matrix" # 检测是否为一个方阵
    P, L = np.eye(row_len), np.zeros((row_len,row_len)) #
P 初始化一个 I 矩阵，L矩阵初始化为一个0矩阵

    for i in range(row_len):
        j = np.argmax(abs(U[i:, i])) + i   # 找到当前列的从i
开始的最大值，然后进行交换
        U = change_row(U, i, j)
        P = change_row(P, i, j) # 同样的，需要对P和L 进行行交
换
        L = change_row(L, i, j)
        pivot = U[i,i] # 保存主元
        assert(pivot!=0), "Error! a zero pivot is
encoutered" # 检测是否存在主元为0 的错误
        for k in range(i,row_len):
            L[k,i] = U[k,i]/pivot
        for m in range(i+1,row_len):
            for n in range(i+1,row_len):
                U[m,i] = 0
                U[m,n] -= L[m,i] * U[i, n]

    return P, L, U  # 这里的matrix经过变化，已经处理为U
```

QR分解

分解思路和条件

QR分解的基本条件如下：

1. 列向量无关的$m \times n$矩阵

具体计算思路为： 利用施密特正交化构建正交矩阵$Q \in R^{m \times n}$，具体为从输入矩阵A的n个列$A_{*i} \in R^{m \times 1}$中构建正交基，具体构建方法如下：

$$q_1 = \frac{a_1}{v_1}$$

$$q_k = \frac{a_k - \sum_{i=1} k - 1 < q_i|a_k > q_i}{v_k}$$

$$\text{其中有} v_1 = ||a_1|| \text{且} v_k = ||a_k - \sum_{i=1} k - 1 < q_i|a_k > q_i||$$

分解后矩阵的性质

1. $A = QR$，其中A为输入矩阵
2. Q为正交矩阵，R为上三角矩阵

方程求解

对于原来的方程$Ax = b$， 其中$A = QR$，所以原来的方程可以变为$QRx = b$，进一步推导为:

$$y = Rx = Q^T b(\text{由于} Q \text{是正交矩阵，所以} Q^{-1} = Q^T)$$

$$x = R^{-1} y$$

由于R为上三角矩阵，对于y进行计算不需要再进行高斯消去操作，可以很简便进行计算。

实例

```
[[11. 10.  4. 18. 18.]
 [ 5.  2.  8. 16. 14.]
 [16. 13.  0.  2. 14.]
 [11. 15. 19.  8.  2.]
 [ 9. 11.  8. 17.  8.]]
----------the QR FACTORIZATION RESULTS----------
Q
[[ 0.448 -0.126 -0.098  0.537  0.697]
 [ 0.203 -0.466  0.832  0.117 -0.188]
 [ 0.651 -0.431 -0.383 -0.451 -0.202]
 [ 0.448  0.678  0.371 -0.404  0.198]
 [ 0.366  0.35  -0.117  0.576 -0.631]]
R
[[ 24.576  24.088  14.852  22.42   23.844]
 [  0.      6.226  11.444   0.795 -10.658]
 [ -0.     -0.     12.388  11.781   4.348]
 [  0.      0.      0.     17.174   8.773]
 [  0.      0.     -0.     -0.      2.447]]
original results
[[11. 10.  4. 18. 18.]
 [ 5.  2.  8. 16. 14.]
 [16. 13. -0.  2. 14.]
 [11. 15. 19.  8.  2.]
 [ 9. 11.  8. 17.  8.]]
----------b----------
[ 5. 14. 15. 17. 14.]
----------x: equation results----------
[ 8.307 -5.942  0.465  1.521 -3.122]
```

程序代码

```
def check(matrix):# 判断QR分解的矩阵条件：所有列线性无关，即矩阵秩=列数
    rank = matrix_rank(matrix)
    n=matrix.shape[-1]
    return rank == n
```

```python
def QRFactorization(matrix): # QR 分解
    '''
    :param matrix: 输入矩阵
    :return: 正交矩阵Q，上三角矩阵R
    '''
    assert(check(matrix)), "Error! Not all columns in matrix are linearly independent"
    Q = np.zeros_like(matrix)
    for index, a in enumerate(matrix.T):
        u = np.copy(a)
        for i in range(index):
            u=u-np.dot(np.dot(Q[:, i].T,a), Q[:, i]) #减去分量
        norm_factor=norm(u) #归一化
        Q[:, index]=u/norm_factor
    R = np.dot(Q.T,matrix)
    return Q, R
```

Householder 约减

分解思路和条件

具体计算思路为：利用householder变换构建正交矩阵$Q \in R^{m\times n}$，具体为从输入矩阵A的n个列$A_{*i} \in R^{m\times 1}$中构建正交基，其中householder变换构造如下：

$$u = a - ||a||_2 \times e_1$$

$$R = I - 2\frac{uu^T}{u^Tu}$$

$$\text{其中的}a\text{为每个子矩阵的第一列}$$

对于所有的R 进行累计，得到

$$P = R_{n-1} \dots R_1$$

分解后矩阵的性质

1. $PA = T$，其中A为输入矩阵
2. P为正交矩阵，T为上三角矩阵

方程求解

对于原来的方程$Ax = b$， 其中$PA = T$，所以原来的方程可以变为$PAx = Pb = Tx$，进一步推导为:

$$y = Tx = Pb(\text{由于}P\text{是正交矩阵，所以}P^{-1} = P^T)$$

$$x = T^{-1}y$$

由于T为上三角矩阵，对于y进行计算不需要再进行高斯消去操作，可以很简便进行计算。

实例

```
[[11. 10.  4. 18. 18.]
 [ 5.  2.  8. 16. 14.]
 [16. 13.  0.  2. 14.]
 [11. 15. 19.  8.  2.]
 [ 9. 11.  8. 17.  8.]]
----------the HOUSEHOLDER REDUCTION RESULTS----------
P
[[ 0.448  0.203  0.651  0.448  0.366]
 [-0.126 -0.466 -0.431  0.678  0.35 ]
 [-0.098  0.832 -0.383  0.371 -0.117]
 [ 0.537  0.117 -0.451 -0.404  0.576]
 [ 0.697 -0.188 -0.202  0.198 -0.631]]
T
[[ 24.576  24.088  14.852  22.42   23.844]
 [ -0.      6.226  11.444   0.795 -10.658]
 [ -0.     -0.     12.388  11.781   4.348]
 [  0.      0.     -0.     17.174   8.773]
 [  0.      0.     -0.      0.      2.447]]
original results
[[11. 10.  4. 18. 18.]
 [ 5.  2.  8. 16. 14.]
 [16. 13. -0.  2. 14.]
 [11. 15. 19.  8.  2.]
 [ 9. 11.  8. 17.  8.]]
----------b----------
[ 5. 14. 15. 17. 14.]
----------x: equation results----------
[ 8.307 -5.942  0.465  1.521 -3.122]
```

程序代码

```
def HouseholderReduction(matrix):# Householder 约减 (为了简便，这里只考虑实数，所以设u为1)
    '''
    :param matrix: 输入矩阵
    :return: 正交矩阵P， 上三角矩阵T
    '''
    row_len=matrix.shape[0]
    P = np.identity(row_len)
    T = np.copy(matrix)
    for index in range(row_len - 1):
        a = T[index:, index]
        u = np.copy(a)
        norm_factor=norm(a) # 归一化因子
        u[0] -= norm_factor # 矩阵列向量-归一化因子*单位向量(变换构造第一二步)
        u = u.reshape(-1, 1)
        R=2.0*(np.dot(u, u.T))/(u.T.dot(u))
        I = np.identity(row_len)
        I[index:, index:] -= R # 得到对应的R 变为 I-2uu.T/u.Tu
        T = np.dot(I, T)
        P = np.dot(I, P)

    return P, T
```

Givens 约减

分解思路和条件

具体计算思路为：利用givens变换构建正交矩阵$Q \in R^{m\times n}$，具体为从输入矩阵A的n个列$A_{*i} \in R^{m\times 1}$中构建正交基，其中旋转矩阵变换构造如下：

$$\text{对于向量}\, x = (x_1\ x_2\ \dots\ x_n)^T$$
$$\text{当实现}\, i, j \,\text{两行的旋转时，对应的旋转矩阵为}$$
$$\begin{bmatrix} 1 & & \vdots & & \vdots & & \\ & \ddots & \vdots & & \vdots & & \\ \cdots & \cdots & c & \cdots & s & \cdots & \cdots \\ & & \vdots & & \vdots & & \\ \cdots & \cdots & -s & \cdots & c & \cdots & \cdots \\ & & \vdots & & \vdots & \ddots & \\ & & \vdots & & \vdots & & 1 \end{bmatrix}$$

其中分别为第i, j行进行处理。

分解后矩阵的性质

1. $PA = T$，其中A为输入矩阵
2. P为正交矩阵，T为上三角矩阵

方程求解

对于原来的方程$Ax = b$，其中$PA = T$，所以原来的方程可以变为$PAx = Pb = Tx$，进一步推导为:

$$y = Tx = Pb(\text{由于}\, P \,\text{是正交矩阵，所以}\, P^{-1} = P^T)$$
$$x = T^{-1}y$$

由于T为上三角矩阵，对于y进行计算不需要再进行高斯消去操作，可以很简便进行计算。

实例

```
[[11. 10.  4. 18. 18.]
 [ 5.  2.  8. 16. 14.]
 [16. 13.  0.  2. 14.]
 [11. 15. 19.  8.  2.]
 [ 9. 11.  8. 17.  8.]]
----------the GIVENS REDUCTION RESULTS----------
P
[[ 0.448  0.203  0.651  0.448  0.366]
 [-0.126 -0.466 -0.431  0.678  0.35 ]
 [-0.098  0.832 -0.383  0.371 -0.117]
 [ 0.537  0.117 -0.451 -0.404  0.576]
 [ 0.697 -0.188 -0.202  0.198 -0.631]]
T
[[ 24.576  24.088  14.852  22.42   23.844]
 [ -0.      6.226  11.444   0.795 -10.658]
 [ -0.     -0.     12.388  11.781   4.348]
 [ -0.      0.      0.     17.174   8.773]
 [ -0.      0.      0.     -0.      2.447]]
original results
[[11. 10.  4. 18. 18.]
 [ 5.  2.  8. 16. 14.]
 [16. 13. -0.  2. 14.]
 [11. 15. 19.  8.  2.]
 [ 9. 11.  8. 17.  8.]]
----------b----------
[ 5. 14. 15. 17. 14.]
----------x: equation results----------
[ 8.307 -5.942  0.465  1.521 -3.122]
```

程序代码

```python
def GivensReduction(matrix): # givens 分解
    '''
    :param matrix: 输入矩阵
    :return: 正交矩阵P， 上三角矩阵T
    '''
    row_len = matrix.shape[0]
    col_len = matrix.shape[1]
    P = np.identity(row_len)
    T = np.copy(matrix)
    for i in range(row_len):
        for j in range(col_len-1,i,-1):
            val_a=T[i,i]
            val_b=T[j,i]
            mag = np.sqrt( (val_a ** 2 + val_b ** 2) )
            c = val_a / mag
            s = val_b / mag
            Pi = np.eye(row_len)
            Pi[i, i] = c
            Pi[j, j] = c
            Pi[i, j] = s
            Pi[j, i] = -s
            P = np.dot(Pi,P)
            T = np.dot(Pi,T)

    return P, T
```

URV 分解

分解思路和条件

具体计算思路为：将矩阵分解为$A=URV^T$，其中

- U的前r列时$R(A)$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

而实际上，标准正交基的求解可以通过Householder 约减来实现（同QR分解和Givens 约减），故思路较为简单。

分解后矩阵的性质

1. $A=URV^T$，其中A为输入矩阵
2. U和V为正交矩阵，R可以表示为$\begin{pmatrix} C_{r\times r} & 0 \\ 0 & 0 \end{pmatrix}$，其中$r$为矩阵A 的秩

方程求解

对于原来的方程$Ax=b$， 其中$A=URV^T$，所以原来的方程可以变为$URV^Tx=b$

在这种情况下，很难对原方程进行优化，故不考虑在URV分解下的方程求解，只采用QR分解情况下的思路进行方程求解。

实例

```
[[11. 10.  4. 18. 18.]
 [ 5.  2.  8. 16. 14.]
 [16. 13.  0.  2. 14.]
 [11. 15. 19.  8.  2.]
 [ 9. 11.  8. 17.  8.]]
----------the URV FACTORIZATION RESULTS----------
U
[[ 0.448 -0.126 -0.098  0.537  0.697]
 [ 0.203 -0.466  0.832  0.117 -0.188]
 [ 0.651 -0.431 -0.383 -0.451 -0.202]
 [ 0.448  0.678  0.371 -0.404  0.198]
 [ 0.366  0.35  -0.117  0.576 -0.631]]
R
[[49.759  0.    -0.     0.    -0.   ]
 [ 1.68  16.767 -0.    -0.    -0.   ]
 [11.089  5.139 12.719 -0.    -0.   ]
 [11.942 -5.96  10.902  8.656  0.   ]
 [ 1.173 -1.673  0.49  -0.907  0.867]]
V
[[ 0.494 -0.049 -0.411 -0.198 -0.739]
 [ 0.484  0.323 -0.552  0.25   0.542]
 [ 0.298  0.653  0.45  -0.529  0.048]
 [ 0.451  0.002  0.533  0.693 -0.181]
 [ 0.479 -0.684  0.2   -0.37   0.354]]
original results
[[11. 10.  4. 18. 18.]
 [ 5.  2.  8. 16. 14.]
 [16. 13.  0.  2. 14.]
 [11. 15. 19.  8.  2.]
 [ 9. 11.  8. 17.  8.]]
```

程序代码

```
def GivensReduction(matrix): # givens 分解
    '''
```

```
    :param matrix: 输入矩阵
    :return: 正交矩阵P， 上三角矩阵T
    '''
    row_len = matrix.shape[0]
    col_len = matrix.shape[1]
    P = np.identity(row_len)
    T = np.copy(matrix)
    for i in range(row_len):
        for j in range(col_len-1,i,-1):
            val_a=T[i,i]
            val_b=T[j,i]
            mag = np.sqrt( (val_a ** 2 + val_b ** 2) )
            c = val_a / mag
            s = val_b / mag
            Pi = np.eye(row_len)
            Pi[i, i] = c
            Pi[j, j] = c
            Pi[i, j] = s
            Pi[j, i] = -s
            P = np.dot(Pi,P)
            T = np.dot(Pi,T)

    return P, T
```

方程求解

在上面已经总结了不同分解下的方程求解，这里不再给出具体计算的说明，只给出相应的代码：

```
import numpy as np

def lowertrinv(L): # 由于L矩阵的对角线元素均为1，所以直接利用其来实现约减 (下三角矩阵)
    N = L.shape[0]
    Linv = np.eye(N)
    matrix = np.copy(L)
    for i in range(N-1):
        for j in range(i+1, N):
            Linv[j, :] -= matrix[j,i] * Linv[i, :]

    return Linv

def uppertrinv(U): # 由于U的对角元素并不为1，所以要先做归一化处理(上三角矩阵求逆)
    N = U.shape[0]
    Uinv = np.eye(N)
    matrix = np.copy(U)
    for i in range(N):
        Uinv[i, :] = Uinv[i, :]/matrix[i, i]
        matrix[i, :] = matrix[i, :]/matrix[i,i]
    for i in range(N-1, -1, -1):
```

```
        for j in range(i-1, -1, -1):
            Uinv[j, :] -= matrix[j,i] * Uinv[i, :]

    return Uinv

def solve(matrixs, data, factorization_type):
    if(factorization_type=='LU'):
        P, L, U = matrixs
        Linv = lowertrinv(L)
        Uinv = uppertrinv(U)
        y = np.dot(P, data)
        tmp = Linv.dot(y)
        x = Uinv.dot(tmp)
        return x
    elif(factorization_type=='QR'):
        Q, R = matrixs
        tmp = np.dot(Q.T, data)
        Rinv = uppertrinv(R)
        x = Rinv.dot(tmp)
        return x
    else:
        P, T = matrixs
        tmp = P.dot(data)
        Tinv = uppertrinv(T)
        x = Tinv.dot(tmp)
        return x
```

行列式计算

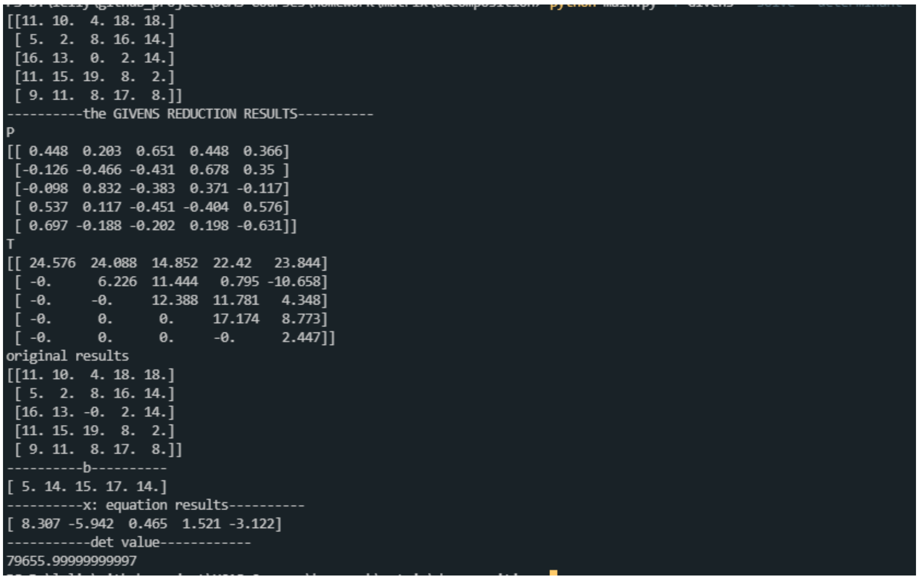
根据行列式计算规则，只有方阵存在行列式的值，并且，对于方阵A，其行列式结果可以表示为：

$det(A) = det(BC)$ 当满足$A = BC$时

且对于上三角矩阵和下三角矩阵，其行列式的结果就等于对角线元素相乘，根据分析，实际上可以使用LU分解来求解A的行列式，并且根据$PA = LU$分解，其中P 的值只能为1 或 -1 ，只需要计算其逆序对的数量。且L 的主对角元素均为1，所以只需要计算U 的主对角元素乘积和P的逆序对，故有：

$$det(A) = \delta(P)det(U)$$
$$\text{其中}\ PA = LU$$

实例



```
[[11. 10.  4. 18. 18.]
 [ 5.  2.  8. 16. 14.]
 [16. 13.  0.  2. 14.]
 [11. 15. 19.  8.  2.]
 [ 9. 11.  8. 17.  8.]]
----------the GIVENS REDUCTION RESULTS----------
P
[[ 0.448  0.203  0.651  0.448  0.366]
 [-0.126 -0.466 -0.431  0.678  0.35 ]
 [-0.098  0.832 -0.383  0.371 -0.117]
 [ 0.537  0.117 -0.451 -0.404  0.576]
 [ 0.697 -0.188 -0.202  0.198 -0.631]]
T
[[ 24.576  24.088  14.852  22.42   23.844]
 [ -0.      6.226  11.444   0.795 -10.658]
 [ -0.     -0.     12.388  11.781   4.348]
 [ -0.      0.      0.     17.174   8.773]
 [ -0.      0.      0.     -0.      2.447]]
original results
[[11. 10.  4. 18. 18.]
 [ 5.  2.  8. 16. 14.]
 [16. 13. -0.  2. 14.]
 [11. 15. 19.  8.  2.]
 [ 9. 11.  8. 17.  8.]]
----------b----------
[ 5. 14. 15. 17. 14.]
----------x: equation results----------
[ 8.307 -5.942  0.465  1.521 -3.122]
-----------det value-----------
79655.99999999997
```

程序代码

```python
import numpy as np

def upperdet(matrix): # 计算上三角矩阵的行列式值，也就是对角线元素相乘
    """
    :param matrix: 输入矩阵
    :param det: 行列式的值
    """
    det = 1
    N = matrix.shape[0]
    for i in range(N):
        det = det * matrix[i,i]
    return det

def ReversedOrder(matrix): # 计算逆序对的数量，并根据结果，输出1或者-1
    """
    :param matrix: 输入矩阵
    :param num: 逆序对数量对应的最终结果
    """
    orders = np.where(matrix==1)[1]
    count = 0
    for i, num in enumerate(orders):
        for j in range(i):
            if num < orders[j]:
                count += 1
    return -1 if count%2 else 1
```

程序主要文件和运行

- decomposition.py 包含了所给出的五种不同的矩阵分解函数
- solution.py 给出了利用矩阵分解结果求解方程的函数
- determinant.py 包含利用LU分解求解行列式的函数
- produce.py 随机生成给定维度的矩阵数据

- main.py 主程序，利用argparse 来打包参数
- run.sh 执行的sh 文件，可以直接通过sh 文件运行程序

一个代表性的执行过程为：

```
python main.py -f LU --solve --determinant
```

其中`-f LU`表示采用LU分解，`--solve`表示激活方程求解，`--determinant`表示激活行列式计算。

主函数代码

```
import numpy as np
import argparse

from decomposition import PLUFactorization,
QRFactorization, HouseholderReduction, GivensReduction,
URVFactorization
from solution import solve
from determinant import upperdet, ReversedOrder

Factorization_Choices =['LU', 'QR', 'Householder',
'Givens', 'URV']

parser = argparse.ArgumentParser(description='Args for
matrix factorization')
parser.add_argument('-f', '--factorization', type=str,
default='URV', choices=Factorization_Choices)
parser.add_argument('-p','--path', type=str,
default='matrix.csv')
parser.add_argument('--solve', default=False,
action='store_true')
parser.add_argument('-d', '--data', type=str,
default='data.csv')
parser.add_argument('--determinant', default=False,
action='store_true')
args = parser.parse_args()

np.set_printoptions(precision=3, suppress=True) #设置小数位
置为3位

def read_data(path):
    matrix = np.loadtxt(path)
    return matrix

if __name__ == '__main__':
    matrix = read_data(args.path)
    print(matrix)
    if(args.factorization == 'LU'):
        matrixs = PLUFactorization(matrix)
```

```
        print('----------the LU FACTORIZATION RESULTS-----
-----')
        print('P')
        print(matrixs[0])
        print('L')
        print(matrixs[1])
        print('U')
        print(matrixs[2])
        print('original results')
        print(np.dot(matrixs[1],matrixs[2]))
    elif(args.factorization == 'QR'):
        matrixs = QRFactorization(matrix)
        print('----------the QR FACTORIZATION RESULTS-----
-----')
        print('Q')
        print(matrixs[0])
        print('R')
        print(matrixs[1])
        print('original results')
        print(np.dot(matrixs[0],matrixs[1]))
    elif(args.factorization == 'Householder'):
        matrixs = HouseholderReduction(matrix)
        print('----------the HOUSEHOLDER REDUCTION
RESULTS----------')
        print('P')
        print(matrixs[0])
        print('T')
        print(matrixs[1])
        print('original results')
        print(np.dot(matrixs[0].T,matrixs[1]))
    elif(args.factorization == 'Givens'):
        matrixs = GivensReduction(matrix)
        print('----------the GIVENS REDUCTION RESULTS-----
-----')
        print('P')
        print(matrixs[0])
        print('T')
        print(matrixs[1])
        print('original results')
        print(np.dot(matrixs[0].T,matrixs[1]))
    else:
        matrixs = URVFactorization(matrix)
        print('----------the URV FACTORIZATION RESULTS----
------')
        print('U')
        print(matrixs[0])
        print('R')
        print(matrixs[1])
        print('V')
        print(matrixs[2])
        print('original results')
```

```
        print(np.dot(np.dot(matrixs[0],matrixs[1]),
matrixs[2].T))

    if args.solve:
        data = read_data(args.data)
        print('----------b----------')
        print(data)
        if(args.factorization == 'URV'):
            matrixs = HouseholderReduction(matrix)
        x = solve(matrixs, data, args.factorization)
        print('----------x: equation results----------')
        print(x)

    if args.determinant:
        assert(matrix.shape[0] == matrix.shape[1]),
"Error! Only square matrix available for determinant
computation"
        if(args.factorization == 'LU'):
            P, _, U = matrixs
        else:
            P, _, U  = PLUFactorization(matrix)

        delta = ReversedOrder(P)
        det = delta * upperdet(U)
        print('-----------det value------------')
        print(det)
```