在命令行运行程序 manipulations.py, 程序使用的是 argparse 形式的命令，所以注意一下相关的指令参数，如图 1 所示：

```
(virtual_2) C:\Users\hp\OneDrive\文档\UCAS-united\矩阵分析\matrix-analysis-main\matrix-analysis-main>python manipulation
s.py --help
usage: manipulations.py [-h] [--manipulation MANIPULATION] [--A A [A ...]]
                        [--nrowsA NROWSA] [--b B [B ...]] [--nrowsb NROWSB]

optional arguments:
  -h, --help            show this help message and exit
  --manipulation MANIPULATION, -man MANIPULATION
                        choose the manipulation,PLU,Gram-
                        Schmidt,Householder,Givens,URV
  --A A [A ...]         the original matrix
  --nrowsA NROWSA       the number of rows of original matrix
  --b B [B ...]         the original equation b (the form Ax=b)
  --nrowsb NROWSB       the number of rows of equation b (the form Ax=b)
```

图 1: 使用–help 之后对于各个参数的说明

其中–manipulation 选择各种操作：LU、Gram-Schmidt、Householder、Givens、URV 这五个里面选择一个，其中 URV 由于分解的结果没有起到实质的简化，这里不用 URV 来求解方程组和行列式，同时 URV 分解里面的 U 使用的是施密特正交化之后的 R(A) 的基

关于矩阵 A 和方程组中 b 的输入，这里是有四个参数对应：

–A：原始 A 矩阵输入，形式是按照行展开，一行一行输入，每一个数字之间使用空格空开

–b：b 的输入，同样是按照行展开，一行一行的输入，每一个数字之间使用空格空开

–nrowsA：A 的行数，用于程序里面将 A 的输入转化成矩阵 A

–nrowsb：b 的行数，用于程序里面将 b 输入转化成矩阵 b

这里要求所有的矩阵都是可逆的，矩阵的秩都是等于矩阵维数的。以 PPT 中的例子进行输入测试程序：

```
A = np.array([
    [1,2,-3,4],
    [4,8,12,-8],
    [2,3,2,1],
    [-3,-1,1,-4]])
b1=np.array([4,16,8,-7])
```

图 2: 程序测试用的例子

PLU:

python manipulations.py –manipulation PLU –nrowsA 4 –A 1 2 -3 4 4 8 12 -8 2 3 2 1 -3 -1 1 -4 –nrowsb 4 –b 4 16 8 -7

结果：

```
(virtual_2) C:\Users\hp\OneDrive\文档\UCAS-united\矩阵分析\matrix-analysis-main\matrix-analysis-main>python manipulations.py --manipulation PLU --nrowsA 4 -
-A 1 2 -3 4 4 8 12 -8 2 3 2 1 -3 -1 1 -4  --nrowsb 4 --b 4 16 8 -7
****manipulation PLU****
U L P
[[  4.   8.  12.  -8.]
 [  0.   5.  10. -10.]
 [  0.   0.  -2.   3.]
 [  0.   0.   0.  -3.]]
 [[ 1.    0.    0.    0.  ]
 [-0.75  1.    0.    0.  ]
 [ 0.5  -0.2   1.    0.  ]
 [ 0.25  0.    3.    1.  ]]
 [[0. 1. 0. 0.]
 [0. 0. 0. 1.]
 [0. 0. 1. 0.]
 [1. 0. 0. 0.]]
LU PA
[[ 4.  8. 12. -8.]
 [-3. -1.  1. -4.]
 [ 2.  3.  2.  1.]
 [ 1.  2. -3.  4.]]
 [[ 4.  8. 12. -8.]
 [-3. -1.  1. -4.]
 [ 2.  3.  2.  1.]
 [ 1.  2. -3.  4.]]
the nonsingular system
[[ 1.  2. -3.  4.]
 [ 4.  8. 12. -8.]
 [ 2.  3.  2.  1.]
 [-3. -1.  1. -4.]] [ 4. 16.  8. -7.]
the solutions:
[1. 1. 1. 1.]
Ax
[ 4. 16.  8. -7.]
det(A)
120.0
validate the det(A)
119.99999999999997
```

图 3: PLU 测试结果

会显示所有的要求，同时还会有验证部分，用来验证输出的正确性

Gram-Schmidt:

python manipulations.py –manipulation Gram-Schmidt –nrowsA 4 –A 1 2 -3 4 4 8 12 -8 2 3 2 1 -3 -1 1 -4 –nrowsb 4 –b 4 16 8 -7

```
(virtual_2) C:\Users\hp\OneDrive\文档\UCAS-united\矩阵分析\matrix-analysis-main\matrix-analysis-main>python manipulations.py --manipulation Gram-Schmidt --n
rowsA 4 --A 1 2 -3 4 4 8 12 -8 2 3 2 1 -3 -1 1 -4  --nrowsb 4 --b 4 16 8 -7
****manipulation Shmidt_factor****
A Q R
[[ 1.  2. -3.  4.]
 [ 4.  8. 12. -8.]
 [ 2.  3.  2.  1.]
 [-3. -1.  1. -4.]]
 [[ 0.18257419  0.14007078 -0.92527712 -0.30151134]
 [ 0.73029674  0.56028312  0.30751857 -0.24120908]
 [ 0.36514837  0.03295783 -0.21771226  0.90453403]
 [-0.54772256  0.81570631 -0.04354245  0.18090681]]
 [[ 5.47722558  7.85068999  8.39841255 -2.5560386 ]
 [ 0.          4.04557371  7.18480708 -7.15184925]
 [ 0.          0.          5.98708726 -6.20479952]
 [ 0.          0.          0.          0.90453403]]
QR Q.T Q
[[ 1.  2. -3.  4.]
 [ 4.  8. 12. -8.]
 [ 2.  3.  2.  1.]
 [-3. -1.  1. -4.]]
[[ 1. -0. -0.  0.]
 [-0.  1.  0.  0.]
 [-0.  0.  1. -0.]
 [ 0.  0. -0.  1.]]
the nonsingular system
[[ 1.  2. -3.  4.]
 [ 4.  8. 12. -8.]
 [ 2.  3.  2.  1.]
 [-3. -1.  1. -4.]] [ 4. 16.  8. -7.]
the solutions:
[1. 1. 1. 1.]
Ax
[ 4. 16.  8. -7.]
det(A) absolute value
119.99999999999993
validate the det(A)
119.99999999999997
```

图 4: Gram-Schmidt 程序测试结果

注意计算 A 行列式的时候，显示的是绝对值，因为难以知道正交矩阵的行列式是 1 还是-1

Householder:

python manipulations.py –manipulation Householder –nrowsA 4 –A 1 2 -3 4 4 8 12 -8 2 3 2 1 -3 -1 1 -4 –nrowsb 4 –b 4 16 8 -7

```
(virtual_2) C:\Users\hp\OneDrive\文档\UCAS-united\矩阵分析\matrix-analysis-main\matrix-analysis-main>python manipulations.py --manipulation Householder --nr
owsA 4 --A 1 2 -3 4 4 8 12 -8 2 3 2 1 -3 -1 1 -4  --nrowsb 4 --b 4 16 8 -7
****manipulation household_red****
A R Q
[[ 1.  2. -3.  4.]
 [ 4.  8. 12. -8.]
 [ 2.  3.  2.  1.]
 [-3. -1.  1. -4.]]
 [[-0.18257419 -0.73029674 -0.36514837  0.54772256]
 [-0.14007078 -0.56028312 -0.03295783 -0.81570631]
 [ 0.92527712 -0.30751857  0.21771226  0.04354245]
 [-0.30151134 -0.24120908  0.90453403  0.18090681]]
 [[-5.47722558 -7.85068999 -8.39841255  2.5560386 ]
 [ 0.         -4.04557371 -7.18480708  7.15184925]
 [ 0.          0.         -5.98708726  6.20479952]
 [ 0.         -0.          0.          0.90453403]]
A=R.T Q , I = R.T R
[[ 1.  2. -3.  4.]
 [ 4.  8. 12. -8.]
 [ 2.  3.  2.  1.]
 [-3. -1.  1. -4.]]
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
the nonsingular system
[[ 1.  2. -3.  4.]
 [ 4.  8. 12. -8.]
 [ 2.  3.  2.  1.]
 [-3. -1.  1. -4.]] [ 4. 16.  8. -7.]
the solutions:
[1. 1. 1. 1.]
Ax
[ 4. 16.  8. -7.]
det(A) absolute value
120.00000000000003
validate the det(A)
119.99999999999997
```

图 5: Householder 程序测试结果

注意计算 A 行列式的时候，显示的是绝对值，因为难以知道正交矩阵的行列式是 1 还是-1

Givens

python manipulations.py –manipulation Givens –nrowsA 4 –A 1 2 -3 4 4 8 12 -8 2 3 2 1 -3 -1 1 -4 –nrowsb 4 –b 4 16 8 -7

```
(virtual_2) C:\Users\hp\OneDrive\文档\UCAS-united\矩阵分析\matrix-analysis-main\matrix-analysis-main>python manipulations.py --manipulation Givens --nrowsA
4 --A 1 2 -3 4 4 8 12 -8 2 3 2 1 -3 -1 1 -4  --nrowsb 4 --b 4 16 8 -7
****manipulation givens_reduction****
A R Q
[[ 1.  2. -3.  4.]
 [ 4.  8. 12. -8.]
 [ 2.  3.  2.  1.]
 [-3. -1.  1. -4.]]
 [[ 0.18257419  0.73029674  0.36514837 -0.54772256]
 [ 0.14007078  0.56028312  0.03295783  0.81570631]
 [-0.92527712  0.30751857 -0.21771226 -0.04354245]
 [-0.30151134 -0.24120908  0.90453403  0.18090681]]
 [[ 5.47722558  7.85068999  8.39841255 -2.5560386 ]
 [-0.          4.04557371  7.18480708 -7.15184925]
 [ 0.          0.          5.98708726 -6.20479952]
 [-0.         -0.          0.          0.90453403]]
A=R.T Q , I = R.T R
[[ 1.  2. -3.  4.]
 [ 4.  8. 12. -8.]
 [ 2.  3.  2.  1.]
 [-3. -1.  1. -4.]]
[[ 1.  0.  0. -0.]
 [ 0.  1.  0. -0.]
 [ 0.  0.  1. -0.]
 [-0. -0. -0.  1.]]
the nonsingular system
[[ 1.  2. -3.  4.]
 [ 4.  8. 12. -8.]
 [ 2.  3.  2.  1.]
 [-3. -1.  1. -4.]] [ 4. 16.  8. -7.]
the solutions:
[1. 1. 1. 1.]
Ax
[ 4. 16.  8. -7.]
det(A) absolute value
119.99999999999991
validate the det(A)
119.99999999999997
```

图 6: Givens 程序测试结果

注意计算 A 行列式的时候，显示的是绝对值，因为难以知道正交矩阵的行列式是 1 还是-1

URV

python manipulations.py –manipulation URV –nrowsA 4 –A 1 2 -3 4 4 8 12 -8 2 3 2 1 -3 -1 1 -4 –nrowsb 4 –b 8 32 16 -14

```
****manipulation URV_factor ****
A U C
[[ 1.  2. -3.  4.]
 [ 4.  8. 12. -8.]
 [ 2.  3.  2.  1.]
 [-3. -1.  1. -4.]]
 [[ 0.18257419  0.14007078 -0.92527712 -0.30151134]
 [ 0.73029674  0.56028312  0.30751857 -0.24120908]
 [ 0.36514837  0.03295783 -0.21771226  0.90453403]
 [-0.54772256  0.81570631 -0.04354245  0.18090681]]
 [[11.2         3.357625   -4.37085975  3.58914187]
 [ 9.49521908 -3.33034623 -0.0087225   4.22925523]
 [ 5.58468383 -4.86397271 -1.03329013  4.29303372]
 [-0.49543369  0.73783412 -0.03938563  0.16363636]]
U C U.T , I = U.T U
[[ 1.  2. -3.  4.]
 [ 4.  8. 12. -8.]
 [ 2.  3.  2.  1.]
 [-3. -1.  1. -4.]]
[[ 1.  0.  0.  0.]
 [ 0.  1.  0. -0.]
 [ 0.  0.  1.  0.]
 [ 0. -0.  0.  1.]]
```

图 7: URV 程序测试结果

注意 URV 使用的是施密特正交化作为计算 U 矩阵 (R(A) 基底) 的方法，最后计算出来的结果由于没有实质上的化简，所以不用来求解方程组和行列式。