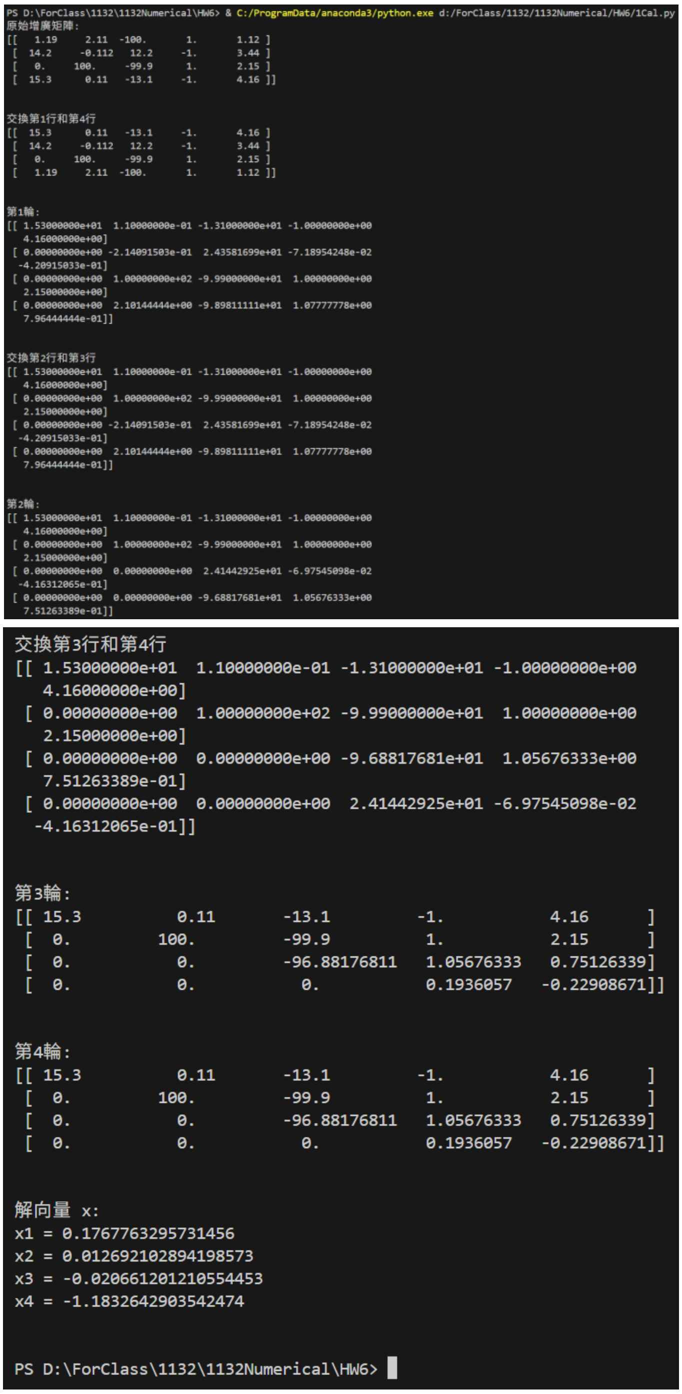
# 數值方法 作業 6

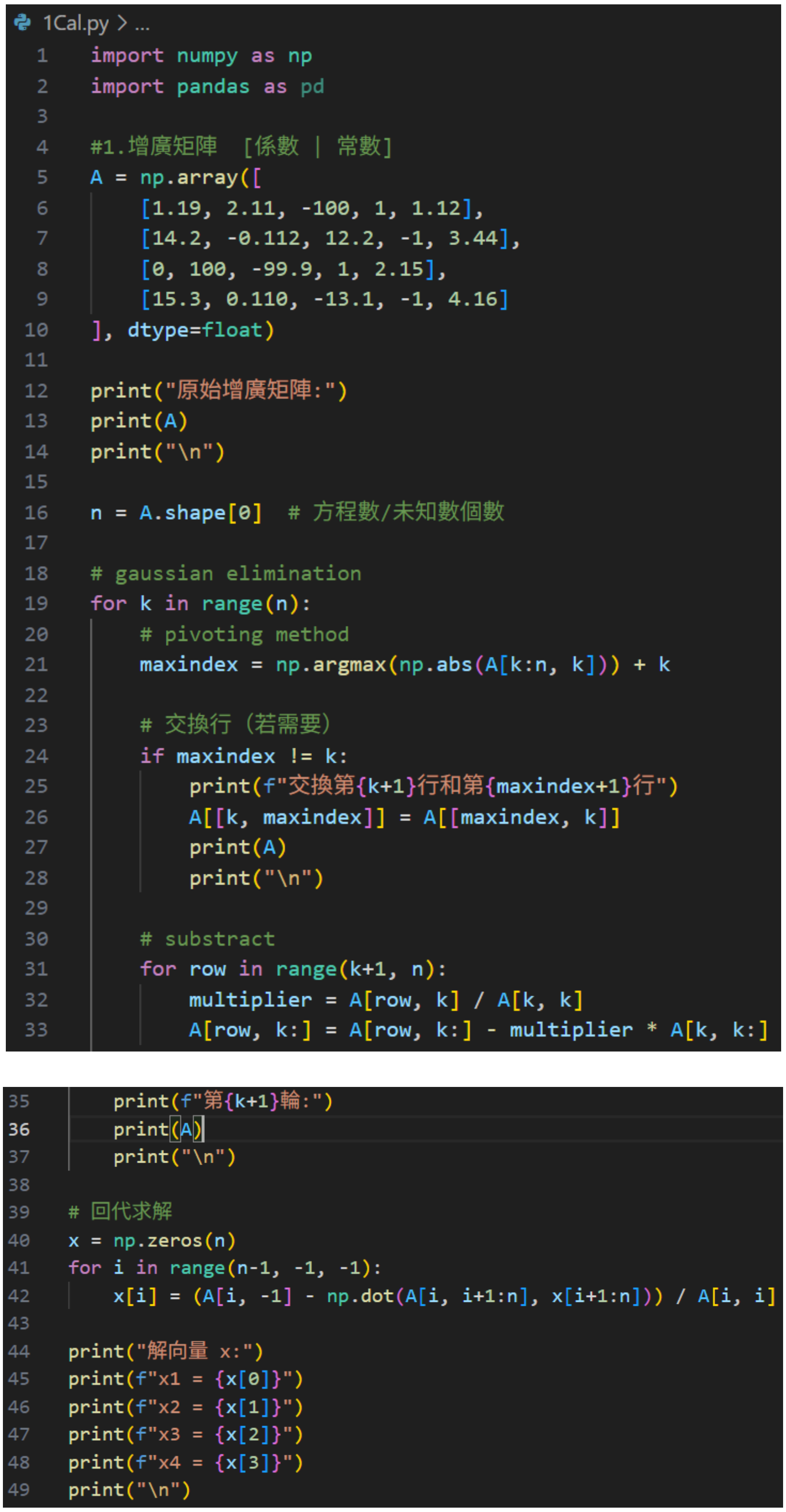
## 工科系 115 E14116401 張瑋哲

### 第一題：



```
PS D:\ForClass\1132\1132Numerical\HW6> & C:/ProgramData/anaconda3/python.exe d:/ForClass/1132/1132Numerical/HW6/1Cal.py
原始增廣矩陣:
[[   1.19     2.11  -100.       1.       1.12 ]
 [  14.2     -0.112   12.2     -1.       3.44 ]
 [   0.     100.      -99.9     1.       2.15 ]
 [  15.3      0.11   -13.1     -1.       4.16 ]]


交換第1行和第4行
[[  15.3      0.11   -13.1     -1.       4.16 ]
 [  14.2     -0.112   12.2     -1.       3.44 ]
 [   0.     100.      -99.9     1.       2.15 ]
 [   1.19     2.11  -100.       1.       1.12 ]]


第1輪:
[[ 1.53000000e+01  1.10000000e-01 -1.31000000e+01 -1.00000000e+00
   4.16000000e+00]
 [ 0.00000000e+00 -2.14091503e-01  2.43581699e+01 -7.18954248e-02
  -4.20915033e-01]
 [ 0.00000000e+00  1.00000000e+02 -9.99000000e+01  1.00000000e+00
   2.15000000e+00]
 [ 0.00000000e+00  2.10144444e+00 -9.89811111e+01  1.07777778e+00
   7.96444444e-01]]


交換第2行和第3行
[[ 1.53000000e+01  1.10000000e-01 -1.31000000e+01 -1.00000000e+00
   4.16000000e+00]
 [ 0.00000000e+00  1.00000000e+02 -9.99000000e+01  1.00000000e+00
   2.15000000e+00]
 [ 0.00000000e+00 -2.14091503e-01  2.43581699e+01 -7.18954248e-02
  -4.20915033e-01]
 [ 0.00000000e+00  2.10144444e+00 -9.89811111e+01  1.07777778e+00
   7.96444444e-01]]


第2輪:
[[ 1.53000000e+01  1.10000000e-01 -1.31000000e+01 -1.00000000e+00
   4.16000000e+00]
 [ 0.00000000e+00  1.00000000e+02 -9.99000000e+01  1.00000000e+00
   2.15000000e+00]
 [ 0.00000000e+00  0.00000000e+00  2.41442925e+01 -6.97545098e-02
  -4.16312065e-01]
 [ 0.00000000e+00  0.00000000e+00 -9.68817681e+01  1.05676333e+00
   7.51263389e-01]]
交換第3行和第4行
[[ 1.53000000e+01  1.10000000e-01 -1.31000000e+01 -1.00000000e+00
   4.16000000e+00]
 [ 0.00000000e+00  1.00000000e+02 -9.99000000e+01  1.00000000e+00
   2.15000000e+00]
 [ 0.00000000e+00  0.00000000e+00 -9.68817681e+01  1.05676333e+00
   7.51263389e-01]
 [ 0.00000000e+00  0.00000000e+00  2.41442925e+01 -6.97545098e-02
  -4.16312065e-01]]


第3輪:
[[ 15.3         0.11       -13.1         -1.           4.16      ]
 [  0.        100.         -99.9          1.           2.15      ]
 [  0.          0.         -96.88176811   1.05676333   0.75126339]
 [  0.          0.           0.           0.1936057   -0.22908671]]


第4輪:
[[ 15.3         0.11       -13.1         -1.           4.16      ]
 [  0.        100.         -99.9          1.           2.15      ]
 [  0.          0.         -96.88176811   1.05676333   0.75126339]
 [  0.          0.           0.           0.1936057   -0.22908671]]


解向量 x:
x1 = 0.1767763295731456
x2 = 0.012692102894198573
x3 = -0.020661201210554453
x4 = -1.1832642903542474


PS D:\ForClass\1132\1132Numerical\HW6> █
```

圖 1、第一題計算結果



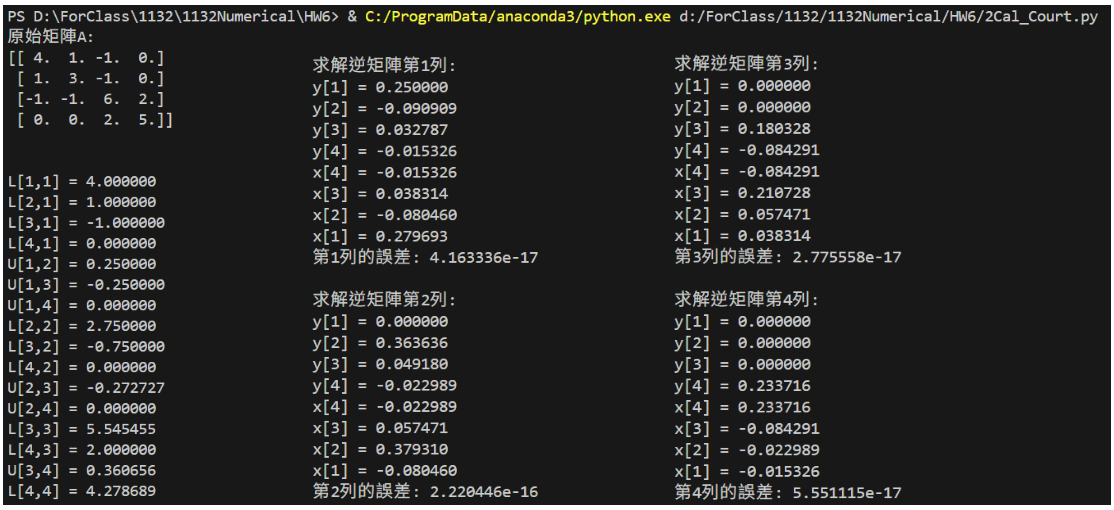
1Cal.py > ...

```python
import numpy as np
import pandas as pd

#1.增廣矩陣  [係數 | 常數]
A = np.array([
    [1.19, 2.11, -100, 1, 1.12],
    [14.2, -0.112, 12.2, -1, 3.44],
    [0, 100, -99.9, 1, 2.15],
    [15.3, 0.110, -13.1, -1, 4.16]
], dtype=float)

print("原始增廣矩陣:")
print(A)
print("\n")

n = A.shape[0]  # 方程數/未知數個數

# gaussian elimination
for k in range(n):
    # pivoting method
    maxindex = np.argmax(np.abs(A[k:n, k])) + k

    # 交換行 (若需要)
    if maxindex != k:
        print(f"交換第{k+1}行和第{maxindex+1}行")
        A[[k, maxindex]] = A[[maxindex, k]]
        print(A)
        print("\n")

    # substract
    for row in range(k+1, n):
        multiplier = A[row, k] / A[k, k]
        A[row, k:] = A[row, k:] - multiplier * A[k, k:]
```

```python
    print(f"第{k+1}輪:")
    print(A)
    print("\n")

# 回代求解
x = np.zeros(n)
for i in range(n-1, -1, -1):
    x[i] = (A[i, -1] - np.dot(A[i, i+1:n], x[i+1:n])) / A[i, i]

print("解向量 x:")
print(f"x1 = {x[0]}")
print(f"x2 = {x[1]}")
print(f"x3 = {x[2]}")
print(f"x4 = {x[3]}")
print("\n")
```

圖 2、第一題計算式

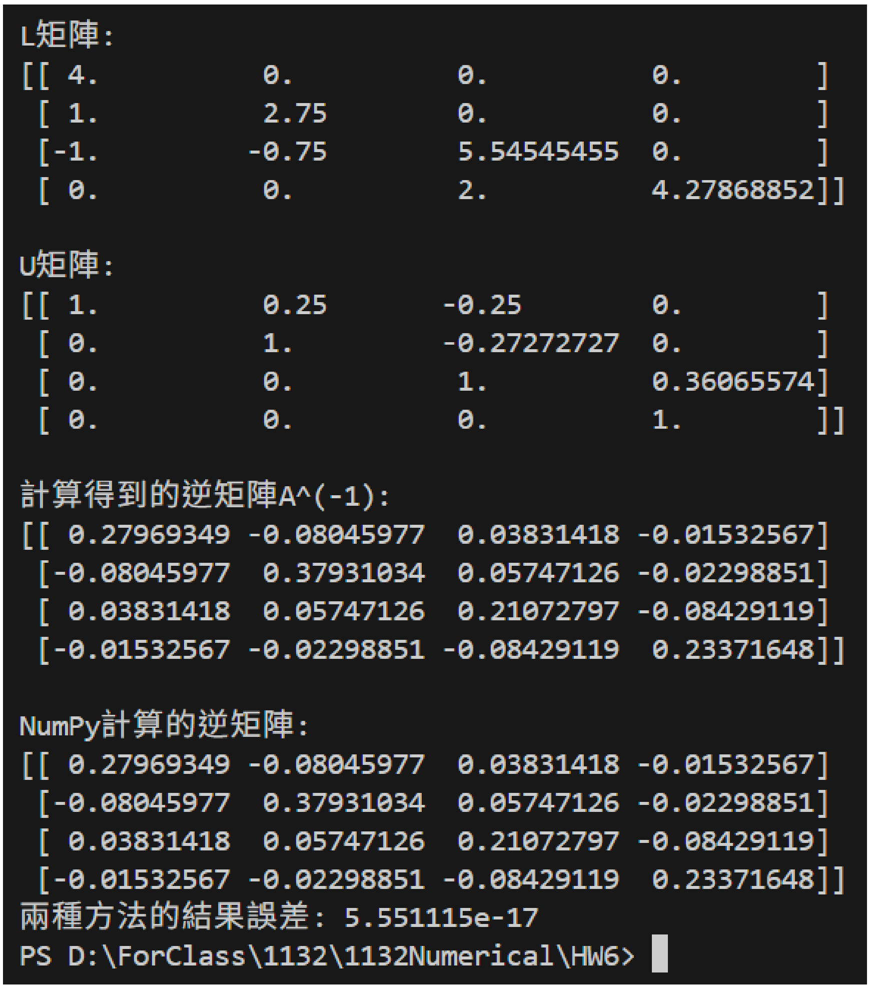
# 第二題：



```
PS D:\ForClass\1132\1132Numerical\HW6> & C:/ProgramData/anaconda3/python.exe d:/ForClass/1132/1132Numerical/HW6/2Cal_Court.py
原始矩陣A:
[[ 4.  1. -1.  0.]
 [ 1.  3. -1.  0.]
 [-1. -1.  6.  2.]
 [ 0.  0.  2.  5.]]


L[1,1] = 4.000000
L[2,1] = 1.000000
L[3,1] = -1.000000
L[4,1] = 0.000000
U[1,2] = 0.250000
U[1,3] = -0.250000
U[1,4] = 0.000000
L[2,2] = 2.750000
L[3,2] = -0.750000
L[4,2] = 0.000000
U[2,3] = -0.272727
U[2,4] = 0.000000
L[3,3] = 5.545455
L[4,3] = 2.000000
U[3,4] = 0.360656
L[4,4] = 4.278689
```

```
求解逆矩陣第1列:
y[1] = 0.250000
y[2] = -0.090909
y[3] = 0.032787
y[4] = -0.015326
x[4] = -0.015326
x[3] = 0.038314
x[2] = -0.080460
x[1] = 0.279693
第1列的誤差: 4.163336e-17

求解逆矩陣第2列:
y[1] = 0.000000
y[2] = 0.363636
y[3] = 0.049180
y[4] = -0.022989
x[4] = -0.022989
x[3] = 0.057471
x[2] = 0.379310
x[1] = -0.080460
第2列的誤差: 2.220446e-16
```

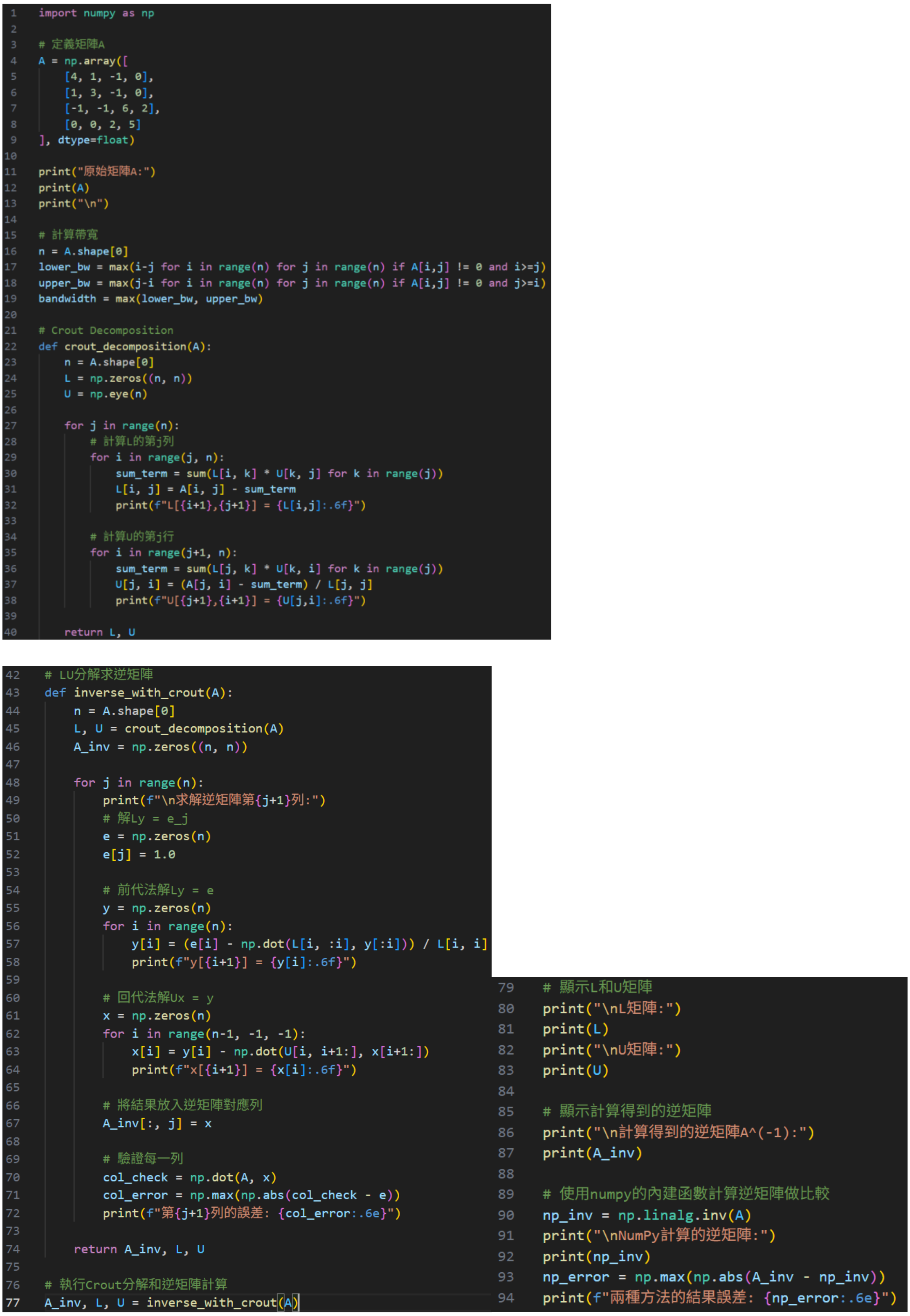
```
求解逆矩陣第3列:
y[1] = 0.000000
y[2] = 0.000000
y[3] = 0.180328
y[4] = -0.084291
x[4] = -0.084291
x[3] = 0.210728
x[2] = 0.057471
x[1] = 0.038314
第3列的誤差: 2.775558e-17

求解逆矩陣第4列:
y[1] = 0.000000
y[2] = 0.000000
y[3] = 0.000000
y[4] = 0.233716
x[4] = 0.233716
x[3] = -0.084291
x[2] = -0.022989
x[1] = -0.015326
第4列的誤差: 5.551115e-17
```



```
L矩陣:
[[ 4.          0.          0.          0.        ]
 [ 1.          2.75        0.          0.        ]
 [-1.         -0.75        5.54545455  0.        ]
 [ 0.          0.          2.          4.27868852]]

U矩陣:
[[ 1.          0.25       -0.25        0.        ]
 [ 0.          1.         -0.27272727  0.        ]
 [ 0.          0.          1.          0.36065574]
 [ 0.          0.          0.          1.        ]]

計算得到的逆矩陣A^(-1):
[[ 0.27969349 -0.08045977  0.03831418 -0.01532567]
 [-0.08045977  0.37931034  0.05747126 -0.02298851]
 [ 0.03831418  0.05747126  0.21072797 -0.08429119]
 [-0.01532567 -0.02298851 -0.08429119  0.23371648]]

NumPy計算的逆矩陣:
[[ 0.27969349 -0.08045977  0.03831418 -0.01532567]
 [-0.08045977  0.37931034  0.05747126 -0.02298851]
 [ 0.03831418  0.05747126  0.21072797 -0.08429119]
 [-0.01532567 -0.02298851 -0.08429119  0.23371648]]
兩種方法的結果誤差: 5.551115e-17
PS D:\ForClass\1132\1132Numerical\HW6> 
```

圖 3、LU 分解逆矩陣計算結果(部分圖疊合節省空間)

先使用 LU 分解求出矩陣，再用回代法 $U\vec{x}=\vec{y}$, $L\vec{y}=\vec{b}$, 計算逆矩陣



```python
import numpy as np

# 定義矩陣A
A = np.array([
    [4, 1, -1, 0],
    [1, 3, -1, 0],
    [-1, -1, 6, 2],
    [0, 0, 2, 5]
], dtype=float)

print("原始矩陣A:")
print(A)
print("\n")

# 計算帶寬
n = A.shape[0]
lower_bw = max(i-j for i in range(n) for j in range(n) if A[i,j] != 0 and i>=j)
upper_bw = max(j-i for i in range(n) for j in range(n) if A[i,j] != 0 and j>=i)
bandwidth = max(lower_bw, upper_bw)

# Crout Decomposition
def crout_decomposition(A):
    n = A.shape[0]
    L = np.zeros((n, n))
    U = np.eye(n)

    for j in range(n):
        # 計算L的第j列
        for i in range(j, n):
            sum_term = sum(L[i, k] * U[k, j] for k in range(j))
            L[i, j] = A[i, j] - sum_term
            print(f"L[{i+1},{j+1}] = {L[i,j]:.6f}")

        # 計算U的第j行
        for i in range(j+1, n):
            sum_term = sum(L[j, k] * U[k, i] for k in range(j))
            U[j, i] = (A[j, i] - sum_term) / L[j, j]
            print(f"U[{j+1},{i+1}] = {U[j,i]:.6f}")

    return L, U
```

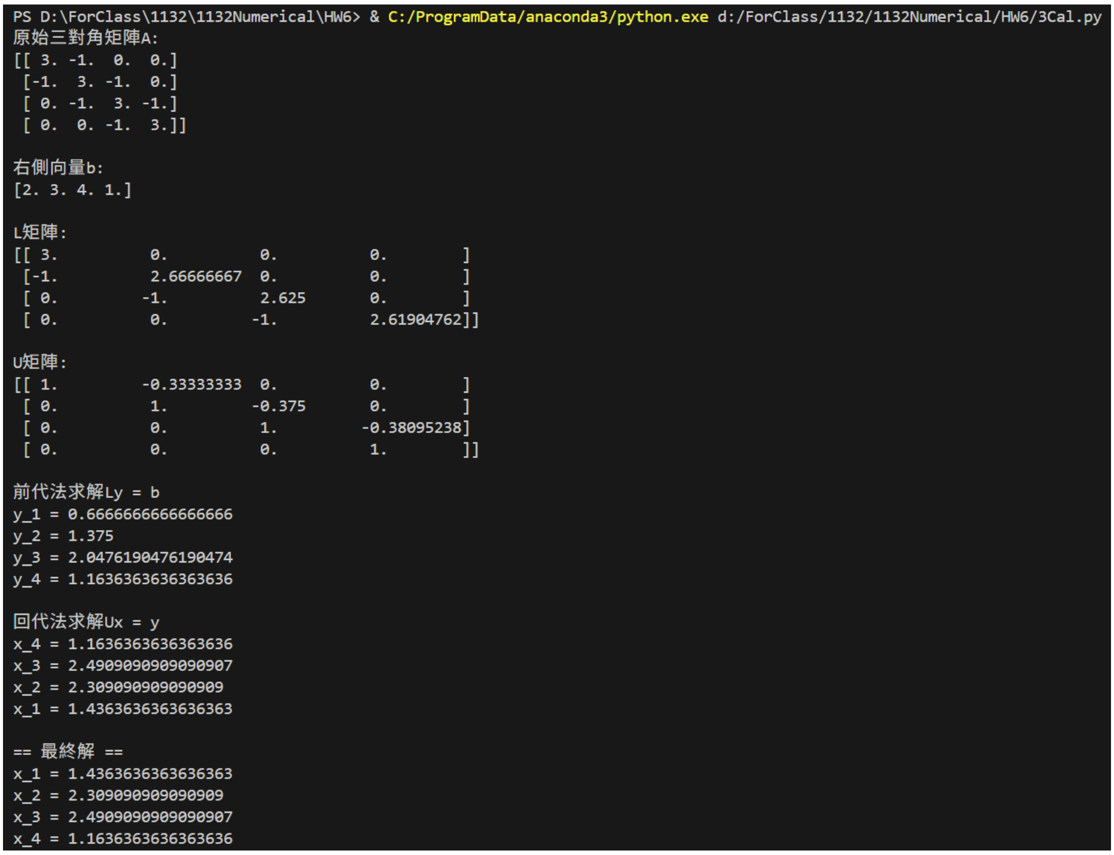
```python
# LU分解求逆矩陣
def inverse_with_crout(A):
    n = A.shape[0]
    L, U = crout_decomposition(A)
    A_inv = np.zeros((n, n))

    for j in range(n):
        print(f"\n求解逆矩陣第{j+1}列:")
        # 解Ly = e_j
        e = np.zeros(n)
        e[j] = 1.0

        # 前代法解Ly = e
        y = np.zeros(n)
        for i in range(n):
            y[i] = (e[i] - np.dot(L[i, :i], y[:i])) / L[i, i]
            print(f"y[{i+1}] = {y[i]:.6f}")

        # 回代法解Ux = y
        x = np.zeros(n)
        for i in range(n-1, -1, -1):
            x[i] = y[i] - np.dot(U[i, i+1:], x[i+1:])
            print(f"x[{i+1}] = {x[i]:.6f}")

        # 將結果放入逆矩陣對應列
        A_inv[:, j] = x

        # 驗證每一列
        col_check = np.dot(A, x)
        col_error = np.max(np.abs(col_check - e))
        print(f"第{j+1}列的誤差: {col_error:.6e}")

    return A_inv, L, U

# 執行Crout分解和逆矩陣計算
A_inv, L, U = inverse_with_crout(A)
```

```python
# 顯示L和U矩陣
print("\nL矩陣:")
print(L)
print("\nU矩陣:")
print(U)

# 顯示計算得到的逆矩陣
print("\n計算得到的逆矩陣A^(-1):")
print(A_inv)

# 使用numpy的內建函數計算逆矩陣做比較
np_inv = np.linalg.inv(A)
print("\nNumPy計算的逆矩陣:")
print(np_inv)
np_error = np.max(np.abs(A_inv - np_inv))
print(f"兩種方法的結果誤差: {np_error:.6e}")
```

圖 4、LU 分解逆矩陣計算過程

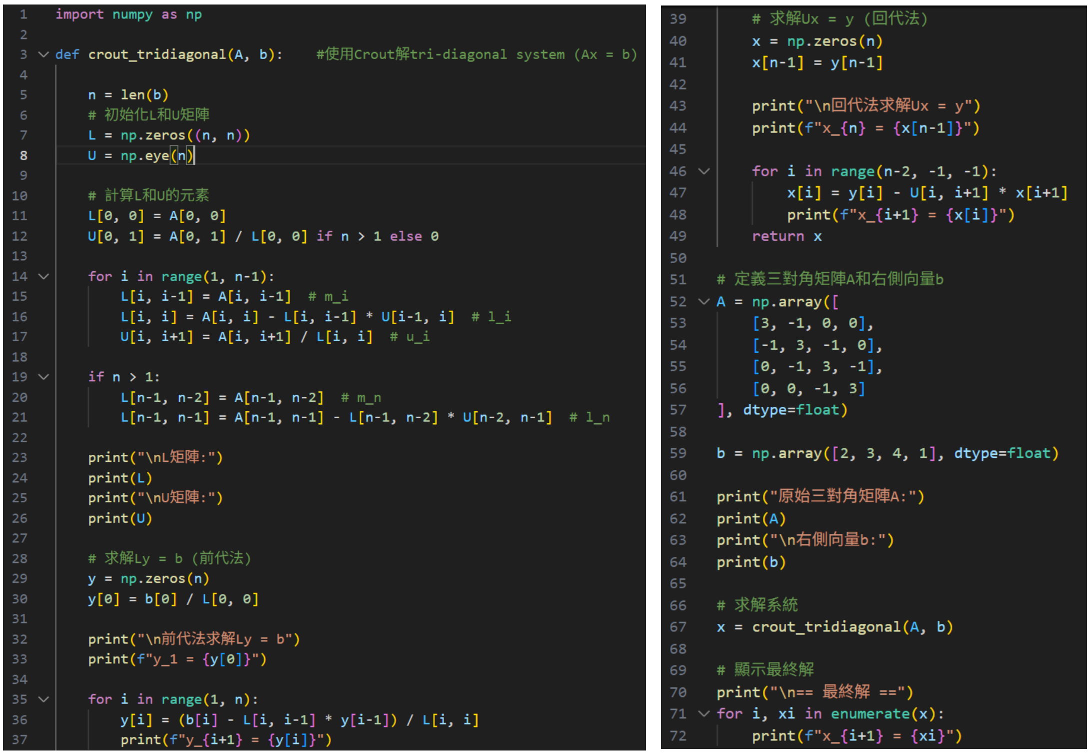
先使用 LU 分解求出矩陣，再用回代法 $U\vec{x}=\vec{y}$, $L\vec{y}=\vec{b}$, 計算逆矩陣

# 第三題：



```
PS D:\ForClass\1132\1132Numerical\HW6> & C:/ProgramData/anaconda3/python.exe d:/ForClass/1132/1132Numerical/HW6/3Cal.py
原始三對角矩陣A:
[[ 3. -1.  0.  0.]
 [-1.  3. -1.  0.]
 [ 0. -1.  3. -1.]
 [ 0.  0. -1.  3.]]

右側向量b:
[2. 3. 4. 1.]

L矩陣:
[[ 3.          0.          0.          0.        ]
 [-1.          2.66666667  0.          0.        ]
 [ 0.         -1.          2.625       0.        ]
 [ 0.          0.         -1.          2.61904762]]

U矩陣:
[[ 1.         -0.33333333  0.          0.        ]
 [ 0.          1.         -0.375       0.        ]
 [ 0.          0.          1.         -0.38095238]
 [ 0.          0.          0.          1.        ]]

前代法求解Ly = b
y_1 = 0.6666666666666666
y_2 = 1.375
y_3 = 2.0476190476190474
y_4 = 1.1636363636363636

回代法求解Ux = y
x_4 = 1.1636363636363636
x_3 = 2.4909090909090907
x_2 = 2.309090909090909
x_1 = 1.4363636363636363

== 最終解 ==
x_1 = 1.4363636363636363
x_2 = 2.309090909090909
x_3 = 2.4909090909090907
x_4 = 1.1636363636363636
```

圖 5、LU 分解後計算過程



```python
import numpy as np

def crout_tridiagonal(A, b):    #使用Crout解tri-diagonal system (Ax = b)

    n = len(b)
    # 初始化L和U矩陣
    L = np.zeros((n, n))
    U = np.eye(n)

    # 計算L和U的元素
    L[0, 0] = A[0, 0]
    U[0, 1] = A[0, 1] / L[0, 0] if n > 1 else 0

    for i in range(1, n-1):
        L[i, i-1] = A[i, i-1]  # m_i
        L[i, i] = A[i, i] - L[i, i-1] * U[i-1, i]  # l_i
        U[i, i+1] = A[i, i+1] / L[i, i]  # u_i

    if n > 1:
        L[n-1, n-2] = A[n-1, n-2]  # m_n
        L[n-1, n-1] = A[n-1, n-1] - L[n-1, n-2] * U[n-2, n-1]  # l_n

    print("\nL矩陣:")
    print(L)
    print("\nU矩陣:")
    print(U)

    # 求解Ly = b (前代法)
    y = np.zeros(n)
    y[0] = b[0] / L[0, 0]

    print("\n前代法求解Ly = b")
    print(f"y_1 = {y[0]}")

    for i in range(1, n):
        y[i] = (b[i] - L[i, i-1] * y[i-1]) / L[i, i]
        print(f"y_{i+1} = {y[i]}")
```

```python
    # 求解Ux = y (回代法)
    x = np.zeros(n)
    x[n-1] = y[n-1]

    print("\n回代法求解Ux = y")
    print(f"x_{n} = {x[n-1]}")

    for i in range(n-2, -1, -1):
        x[i] = y[i] - U[i, i+1] * x[i+1]
        print(f"x_{i+1} = {x[i]}")
    return x

# 定義三對角矩陣A和右側向量b
A = np.array([
    [3, -1, 0, 0],
    [-1, 3, -1, 0],
    [0, -1, 3, -1],
    [0, 0, -1, 3]
], dtype=float)

b = np.array([2, 3, 4, 1], dtype=float)

print("原始三對角矩陣A:")
print(A)
print("\n右側向量b:")
print(b)

# 求解系統
x = crout_tridiagonal(A, b)

# 顯示最終解
print("\n== 最終解 ==")
for i, xi in enumerate(x):
    print(f"x_{i+1} = {xi}")
```

圖 6、LU 分解後計算程式碼