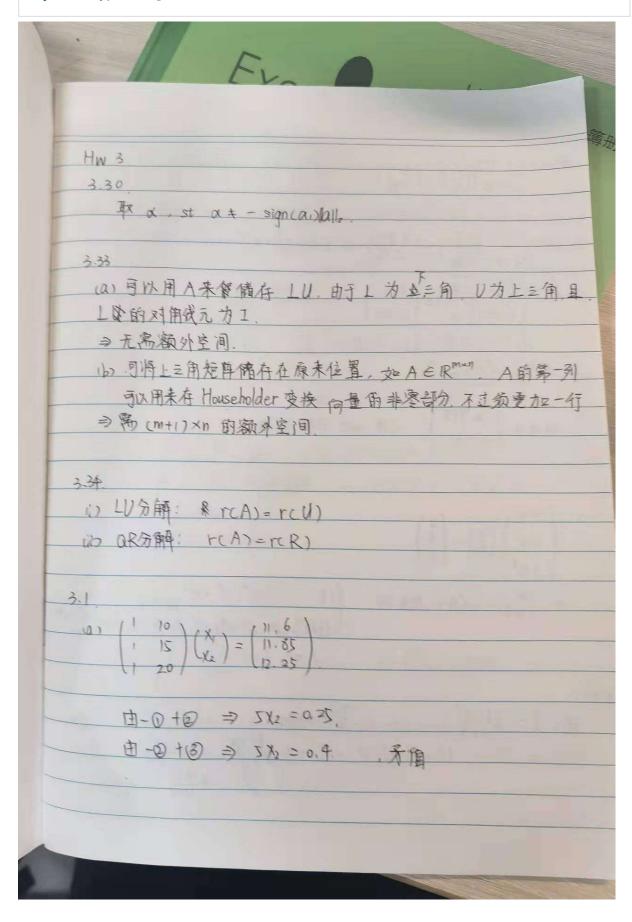
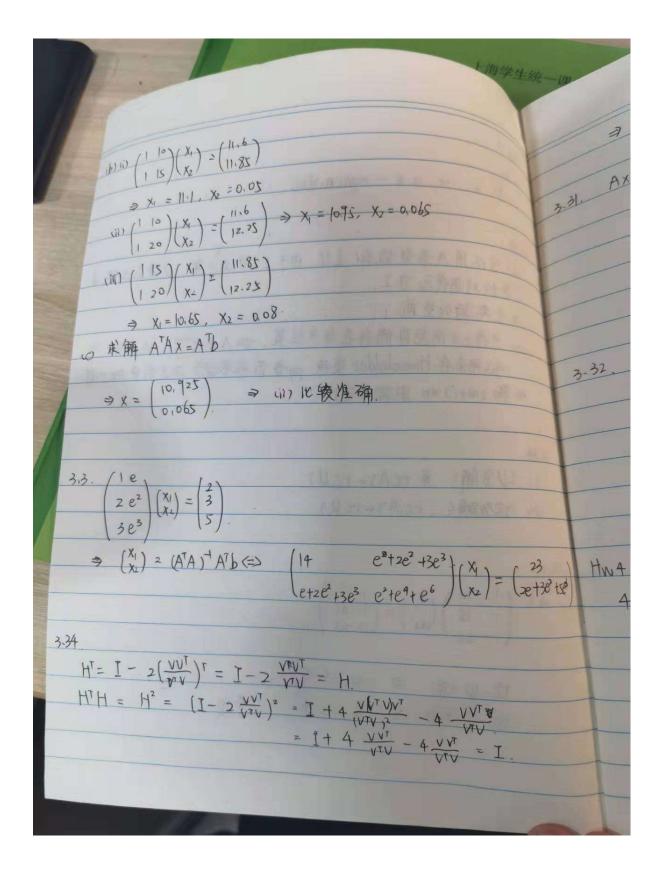
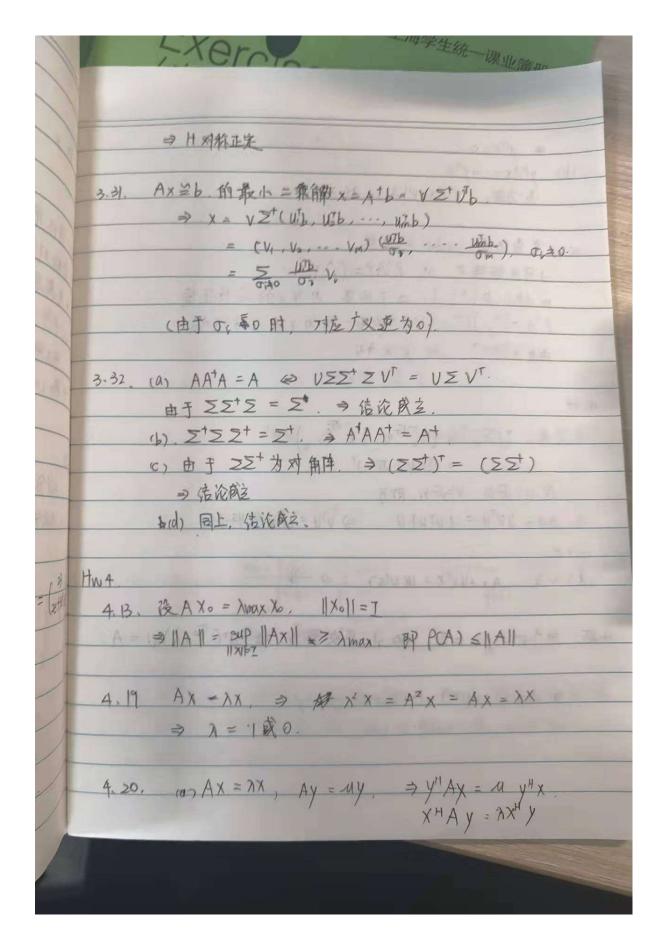
In []:

import numpy as np
import scipy
import scipy.linalg







T1

Computer problem (in C or C++): Using Gaussian elimination to achieve the LU decomposition with and without a column pivoting; Using the two LU decomposition algorithm to solve linear systems in which the coefficient matrix is (1) general nonsingular matrix; (2) positive definite

matrix; (3) diagonally dominant matrix. Compare the numerical accuracy for the two algorithms. The size of the matrices should be greater than 1000.

```
In [ ]:
          def Q_i(Q_min, i, j, k):
              if i < k or j < k:
                  return float(i == j)
              else:
                  return Q_min[i-k][j-k]
          def QR householder (A):
              n = A. shape[0]
              # Set R equal to A, and create Q as a zero matrix of the same size
              R = A \cdot copy()
              Q = np.zeros((n, n))
              # The Householder procedure
              for k in range (n-1):
                  I = np. eye(n)
                  x = R[k:, k]
                  e = I[k:, k]
                  alpha = np. sign(x[0]) * np. linalg. norm(x)
                  # Using anonymous functions, we create u and v
                  u = np. array(list(map(lambda p, q: p + alpha * q, x, e)))
                  # print(u)
                  norm u = np.linalg.norm(u)
                  v = np.array(list(map(lambda p: p/norm_u, u)))
                  # Create the Q minor matrix
                  Q_{min} = np. array([[float(i==j) - 2.0 * v[i] * v[j] for i in range(n-k)] for j in range(n-k)] for j
                  # "Pad out" the Q minor matrix with elements from the identity
                  Q_t = np. array([[ Q_i(Q_min, i, j, k) for i in range(n)] for j in range(n)])
                  if k == 0:
                       Q = Q_t
                       R = Q_t@A
                  else:
                       Q = Q t@Q
                       R = Q t@R
              # Since Q is defined as the product of transposes of Q t,
              # we need to take the transpose upon returning it
              return Q.T, R
```

```
continue
for j in range(k, A. shape[1]):
    gamma_j = np. transpose(v_k)@A[:, j]
    A[:, j] = A[:, j] - np. reshape((2*gamma_j/beta_k)*v_k, np. shape(A)[0])
H = np. eye(A. shape[0]) - 2 / beta_k * v_k@np. transpose(v_k)
A_star = P_k@A_star
A_star = H@A_star
Q = Q@np. transpose(P_k)
Q = Q@np. transpose(H)
return Q , A
```

验证

```
In [ ]:
         A = np.array([[12, -51, 4], [6, 167, -68], [-4, 24, -41]])
         Q, R = QR householder (A)
In [ ]:
         print("Q, R分别为: ", Q, R)
        Q, R分别为: [[-0.85714286 0.39428571 0.33142857]
         [-0.42857143 -0.90285714 -0.03428571]
         [0.28571429 -0.17142857 \quad 0.94285714]] \quad [[-1.40000000e+01 -2.10000000e+01 \quad 1.40000000e+01]
         +017
         [-5.57673565e-16 -1.75000000e+02 7.00000000e+01]
          [-5.08994556e-16 -7.64989650e-16 -3.50000000e+01]
In [ ]:
         q, r = scipy.linalg.qr(A)
         print("Q, R与真实的Q, R对比为: ", Q-q, R-r)
        Q,R与真实的Q,R对比为: [[ 2.22044605e-16 -1.66533454e-16 -5.55111512e-17]
         [ 5.55111512e-17 2.22044605e-16 -4.85722573e-17]
          [-5.55111512e-17 -2.77555756e-17 0.00000000e+00]] \ [[\ 1.77635684e-15\ 0.00000000e+00]]
        -3.55271368e-15]
          [-5.57673565e-16.5.68434189e-14.-2.84217094e-14]
          [-5.08994556e-16 -7.64989650e-16 0.00000000e+00]]
```