In [ ]:
```
import numpy as np
from scipy.linalg import lu
import matplotlib.pyplot as plt
```

2.45

2.48

$$A = LU$$

where $\boldsymbol{L}$ is unit lower triangular and $\boldsymbol{U}$ is upper triangular. Given such a factorization, the linear system $Ax = b$ can be written as $LUx = b$ and hence can be solved by first solving the lower triangular system $Ly = b$ by forward-substitution, then the upper triangular system $Ux = y$ by back-substitution.

2.49

(1)

$$LPx = b$$

where now L really is lower triangular. To solve the linear system, we first solve the lower triangular system $\boldsymbol{Ly} = \boldsymbol{Pb}$ by forward-substitution, then $x = P^T y$

(2)

$$PLx = b$$

where now L really is lower triangular. To solve the linear system, we solve the lower triangular system $\boldsymbol{Lx} = \boldsymbol{P}^T\boldsymbol{b}$ by forward-substitution.

2.51

$x = (1.5, 1.5), y = (2, 0)$

$$||x||_1 = 3 > 2 = ||y||_1$$

$$||x||_\infty = 1.5 < 2 = ||y||_\infty$$

2.52

$||A||_1$更容易计算

2.53

(1)

$cond(A) = \|A\|_1 \|A^{-1}\|_1 = 6 \times 0.5 = 3$

(2)

$cond(A) = \|A\|_\infty \|A^{-1}\|_\infty = 6 \times 0.5 = 3$

2.61

计算条件数可得 (a)(d) well-conditioned (b)(c) ill-conditioned

2.77

A 的 cholesky分解为：

$$\begin{bmatrix} 2 & 0 \\ 1 & 1 \end{bmatrix}$$

In [ ]:

```
A = np.array([[4,2],[2,2]])
np.linalg.cholesky(A)
```

Out[ ]:

```
array([[2., 0.],
       [1., 1.]])
```

2.10

(1)

$$P = P_n \cdots P_2 P_1$$

$P^{-1} = (P_n \cdots P_2 P_1)^{-1} = P_1^{-1} P_2^{-1} \cdots P_n^{-1} = P_1 P_2 \cdots P_n = P_1 P_2 \cdots P_n = P_1^T P_2^T \cdots P_n^T =$

(2)

$$PA = P(M_{n-1} P_{n-1} \cdots M_1 P_1)^{-1} U = P(P_1 M_1^{-1} P_2 \cdots P_n M_n^{-1}) U$$

即证明 $P(P_1 M_1^{-1} P_2 \cdots P_n M_n^{-1})$为下三角矩阵。

$$\begin{aligned} P(P_1 M_1^{-1} P_2 \cdots P_n M_n^{-1}) &= P_n \cdots P_2 P_1 P_1 M_1^{-1} P_2 \cdots P_n M_n^{-1}, \\ &= P_n \cdots P_2 M_1^{-1} P_2 \cdots P_n M_n^{-1} \\ &= P_n \cdots P_2 (I + m_1 e_1) P_2 \cdots P_n M_n^{-1}, \end{aligned}$$

由于$P_2(I + m_1 e_1) P_2 = I + m_1' e_1$(由于$P_2$只对第二行以及以后的行做变换,$m_1'$即为对应两个元素交换所得)，考虑$(I + m_1' e_1)(I + m_2 e_2) = I + \sum_{i=1}^2 m_i e_i$,其也为下三角。

长此以往，可得$P(P_1 M_1^{-1} P_2 \cdots P_n M_n^{-1})$也为下三角矩阵。

2.31

- $||x||_A=(x^TAx)^{\frac{1}{2}}=(x^TLL^Tx)=(y^Ty)=\sum_i y_i^2>=0$ (A positive defined), And $||x||_A=0$ if and only if $y=x=0$
- $||\alpha x||_A=(\alpha^2x^TAx)^{\frac{1}{2}}=\alpha(x^TAx)^{\frac{1}{2}}=\alpha||x||_A$
- 只需证明$(x^TAy)^2<=x^TAxy^TAy$,令$A=U^T\Lambda U,\Lambda=diag(a_1,\cdots,a_n)$,则 $Ux=(x_1,\cdots,x_n),Uy=(y_1,\cdots,y_n)$, 于是上式子等于 $(a_1x_1y_1,\cdots,a_nx_ny_n)^2<=(a_1x_1^2+\cdots)(a_1y_1^2+\cdots)$, 而由柯西不等式，上式成立。

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 pivot_matrix(M, j):

    m = M.shape[0]

    id_mat = np.eye(m)
    row = max(range(j,  m),  key=lambda i: abs(M[i][j]))
    # print(row)
    if j != row:
        id_mat[[j,row],:]= id_mat[[row,j],:]
    return id_mat, row

def LU_Decomposition(A,pivot = False):
    N = A.shape[0]
    A_copy = A.copy()

    L = np.eye(N)
    U = np.zeros((N,N))
    P = np.eye(N)
    for i in range(N):
        if pivot:
            tmp_P,row = pivot_matrix(A_copy,i)
            P = tmp_P@P
            A_copy = tmp_P@A_copy
            L[[i,row],i-1] = L[[row,i],i-1]
        L[i:,i] = A_copy[i:,i]/A_copy[i,i]
        # print(i,A[i,i])
        U[i,i:] = A_copy[i,i:]
        A_copy[i:,i:] = A_copy[i:,i:] - L[i:,i][:,None]@U[i,i:][None,:]
        # print(A,L[i:,i].reshape([-1,1]).shape,U[i,i:].shape)
    if pivot:
        return P,L,U
    return L,U
```

(1)非奇异矩阵

In [ ]:
```
A = np.random.rand(1000,1000)
b = np.random.rand(1000,1)
v,w = np.linalg.eig(A)
print("A的特征值为0的个数为：",np.sum((v==0)))
```

A的特征值为0的个数为： 0

In [ ]:
```
L,U = LU_Decomposition(A)
y = np.linalg.solve(L,b)
x = np.linalg.solve(U,y)
error = np.sum(x-np.linalg.inv(A)@b)

print("LU分解的误差为：",error)
```

LU分解的误差为： 2.9181106885262445e-10

In [ ]:
```
P,L,U = LU_Decomposition(A,pivot=True)
y = np.linalg.solve(L,P*b)
x = np.linalg.solve(U,y)
error = np.sum(x-np.linalg.inv(A)@b)

print("PLU分解的误差为：",error)
```

PLU分解的误差为： 4.377166032789367e-10

(2)正定矩阵

In [ ]:
```
A = A.T@A
print("A的特征值小于等于0的个数为：",np.sum((v==0)))
```

A的特征值小于等于0的个数为： 0

In [ ]:
```
L,U = LU_Decomposition(A)
y = np.linalg.solve(L,b)
x = np.linalg.solve(U,y)
error = np.sum(x-np.linalg.inv(A)@b)

print("LU分解的误差为：",error)
```

LU分解的误差为： 1.8817684729818256e-11

In [ ]:
```
P,L,U = LU_Decomposition(A,pivot=True)
y = np.linalg.solve(L,P*b)
x = np.linalg.solve(U,y)
error = np.sum(x-np.linalg.inv(A)@b)

print("PLU分解的误差为：",error)
```

PLU分解的误差为： 2.8226527094727377e-15

(3)对角占优矩阵

In [ ]:
```
A = np.eye(1000)*5
```

In [ ]:
```
L,U = LU_Decomposition(A)
y = np.linalg.solve(L,b)
x = np.linalg.solve(U,y)
```

```python
error = np.sum(x-np.linalg.inv(A)@b)

print("LU分解的误差为：",error)
```

LU分解的误差为： -5.794681141174651e-15

In [ ]:

```python
L,U = LU_Decomposition(A)
y = np.linalg.solve(L,b)
x = np.linalg.solve(U,y)
error = np.sum(x-np.linalg.inv(A)@b)

print("PLU分解的误差为：",error)
```

PLU分解的误差为： -8.692021711761977e-15