In [ ]:

```python
import numpy as np

import scipy

import matplotlib.pyplot as plt

import sympy as sp
```

一般来说,我们在解线性方程组是有两种办法:直接法和迭代法,在本章中,我们将介绍直接法.

直接法根本的思想即使Gauss消元法.\ 即通过初等行列变换将矩阵A变换为上三角矩阵来方便我们求解.\ 在介绍Gauss消元法之前,我们先介绍一些基础的矩阵知识,我们将用代码来实现.

In [ ]:

```python
class matrixOfSteven:
    def __init__(self,a):
        self.array = a
        self.rows = len(a)
        self.columns = len(a[0])

    def __add__(self,other):
        if self.rows != other.rows or self.columns != other.columns:
            raise ValueError("矩阵相加时，行列数必须相同")


        for i in range(self.rows):
            for j in range(self.columns):
                self.array[i,j] += other.array[i,j]

    def __mul__(self,other):
        if self.columns != other.rows:
            raise ValueError("矩阵相乘时，第一矩阵的列数必须等于第二矩阵的行数")

        a = np.zeros((self.rows,other.columns))

        for i in range(self.rows):
            for j in range(self.columns):
                a[i,j] = sum(self.array[i,k] * other.array[k,j] for k in range(self.columns))

        return matrixOfSteven(a)

    def __eq__(self,other):
        for i in range(self.rows):
            for j in range(self.columns):
                if self.array[i,j] != other.array[i,j]:
                    return False
        return True

    def __repr__(self) -> str:
        return str(self.array)

    def T(self):
        a = np.zeros((self.columns,self.rows))
        for i in range(self.rows):
            for j in range(self.columns):
                a[j,i] = self.array[i,j]

        return matrixOfSteven(a)
```

上述例子供演示使用,在实际使用中我们仍然使用计算速度更快且功能更完善的numpy包中的矩阵来实现后续功能

下面我们将介绍Gauss消元法及其实现过程:\ Gauss消元法是线性方程组的求解方法，其原理是把线性方程组转化为一个矩阵，然后通过矩阵的消元过程求解线性方程组的解。\ 首先，我们定义一个函数 `gauss_elimination`，接收一个参数 `A`，表示线性方程组的系数矩阵，返回一个数组 `x`，表示线性方程组的解。\ 函数的实现过程如下：

In [ ]:

```python
def gauss_solve(A_input,b_input):
    A = A_input.copy()
    b = b_input.copy()

    n = len(A)
    m = len(A[0])

    x = np.zeros((n,1))

    if n != m or n != len(b):
        return None

    for i in range(n-1):

        for j in range(i+1,m):
            if A[i][i] == 0:
                print(f"第{i+1}行第{j+1}列元素为0,无法消元")
                return None

            if A[i][i] != 0:
                param = A[j][i]/A[i][i]
                for k in range(i,m):
                    A[j][k] = A[j][k] - param*A[i][k]
                b[j] = b[j] - param*b[i]


    x[n-1,0] = b[n-1] / A[n-1][n-1]

    for i in range(n-2,-1,-1):
        t = 0
        for j in range(i+1,n):
            t = t + A[i][j]*x[j,0]

        x[i,0] = (b[i] - t) / A[i][i]


    return x
```

In [ ]:

```python
A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 10]])
b = np.array([1, 2, 3])
x = np.linalg.solve(A, b)
x1 = gauss_solve(A, b)
print(x)
print(x1)
```

```
[-3.33333333e-01  6.66666667e-01  3.17206578e-17]
[[-0.33333333]
 [ 0.66666667]
 [ 0.        ]]
```

容易发现,我们自己编写的程序在计算精准度上不如numpy库来的精准与快速

In [ ]:

```python
def L_U_decomposition(A,b = [],isCompute = False):

    """
    对矩阵A进行LU分解,若b不为空,则求解线性方程组Ax = b

    Args:
        A (Matrix): 系数矩阵
        b (list, optional): 方程组右侧常数向量. Defaults to [].

    Returns:
        x : 方程组的解
    """

    n = len(A)

    L = np.zeros((n,n))
    U = np.zeros((n,n))

    for i in range(n):
        L[i][i] = 1
        if i == 0:
            U[0][0] = A[0][0]
            for j in range(1,n):
                U[0][j] = A[0][j]
                L[j][0] = A[j][0]/U[0][0]
        else:
            for j in range(i,n):  ##生成U矩阵
                temp = 0
                for k in range(0,i):
                    temp += L[i][k] * U[k][j]
                U[i][j] = A[i][j] - temp
            for j in range(i+1,n): ##生成L矩阵
                temp = 0
                for k in range(0,i):
                    temp += L[j][k] * U[k][i]
                L[j][i] = (A[j][i] - temp)/U[i][i]

    if isCompute:

        Z = scipy.linalg.solve(L,b)
        x = scipy.linalg.solve(U,Z)

    else:
        x = []

    return L,U,x
```

In [ ]:

```python
A = np.array([[1,1,1],[7,4,-1],[2,-2,1]]).astype(float)
b = np.array([1,2,3]).astype(float)
L,U,x = L_U_decomposition(A,b,True)
print("\nL:\n",L,"\nU:\n",U,"\nx:\n",x)
```

```
L:
 [[1.         0.         0.        ]
 [7.         1.         0.        ]
 [2.         1.33333333 1.        ]]
U:
 [[ 1.          1.          1.        ]
 [ 0.         -3.         -8.        ]
 [ 0.          0.          9.66666667]]
x:
 [ 0.65517241 -0.44827586  0.79310345]
```

In [ ]:

```python
def Q_R_decomposition(A):

    Q = np.zeros_like(A)

    cnt = 0
    for a in A.T:
        u = np.copy(a)
        for i in range(cnt):
            u -= np.dot(np.dot(Q[:,i].T,a),Q[:,i])
        e = u / np.linalg.norm(u)
        Q[:,cnt] = e
        cnt += 1
    R = np.dot(Q.T,A)
    return Q,R
```

In [ ]:

```python
q,r = Q_R_decomposition(A)

print("QR分解为:\nQ\n",q,"\nR\n",r)
print(np.dot(q,r))
```

```
QR分解为:
Q
 [[ 0.13608276  0.17492111  0.97513286]
 [ 0.95257934  0.24730226 -0.17729688]
 [ 0.27216553 -0.95301847  0.13297266]]
R
 [[ 7.34846923e+00  3.40206909e+00 -5.44331054e-01]
 [ 6.66133815e-16  3.07016708e+00 -1.02539962e+00]
 [ 1.66533454e-16 -5.55111512e-17  1.28540240e+00]]
[[ 1.  1.  1.]
 [ 7.  4. -1.]
 [ 2. -2.  1.]]
```

In [ ]:

```python
def Cholesky_decomposition(A):

    """
    对矩阵做Chpolsky分解
    """


    w = A.shape[0]

    L = np.zeros((w,w))

    for i in range(w):
        L[i,i] = 1

    D = np.zeros((w,w))

    for i in range(w):
        D[i,i] = A[i,i] - np.dot(np.dot(L[i,:i],D[:i,:i]),L[i,:i].T)
        for j in range(i+1,w):
            L[j,i] = (A[j,i] - np.dot(np.dot(L[j,:i],D[:i,:i]),L[i,:i].T))/D[i,i]

    return L,D
```

In [ ]:

```python
L,D = Cholesky_decomposition(np.dot(A.T,A))

print("L:\n",L,"\nD:\n",D)
```

```
L:
 [[ 1.          0.          0.        ]
 [ 0.46296296  1.          0.        ]
 [-0.07407407 -0.33398821  1.        ]]
D:
 [[54.          0.          0.        ]
 [ 0.          9.42592593  0.        ]
 [ 0.          0.          1.65225933]]
```

In [ ]:

```python
def column_main_gauss_elimination(A_in:np.array, b_in:np.array) -> np.array:

    """
    Summeration of the column-main Gauss elimination method for solving linear equations:
    采用列主元高斯消元法求解线性方程组

    Args:
        A_in (Matrix): 要求方阵
        b_in (Matrix): 要求长度与A_in行数相同的向量

    Returns:
        Matrix[]: 求解线性方程组的解向量

    """

    A = A_in.copy()

    b = b_in.copy()

    m = len(A)
    n = len(A[0])

    if m!=n:
        return "A is not a square matrix"

    x = np.zeros((n,1))

    i = 0
    while i < m-1:

        column_max = abs(A[i][i])
        column_max_index = i

        for j in range(i+1, m):

            if abs(A[j][i]) > abs(column_max):
                column_max = A[j][i]
                column_max_index = j

        if column_max == 0:
            return "A is singular"

        if i != j:
            for k in range(i,m):

                A[i][k], A[column_max_index][k] = A[column_max_index][k], A[i][k]

            b[i], b[column_max_index] = b[column_max_index], b[i]


        for j in range(i+1,m):

            param = A[j][i]/A[i][i]

            for k in range(i,m):
                A[j][k] = A[j][k] - param*A[i][k]
            b[j] = b[j] - param*b[i]


        i+=1


    x[n-1,0] = b[n-1] / A[n-1][n-1]

    for i in range(m-2,-1,-1):
        t = 0
        for j in range(i+1,n):
            t = t + A[i][j]*x[j,0]

        x[i,0] = (b[i] - t) / A[i][i]

    return x
```

In [ ]:

```python
print(A,b)

L2 = column_main_gauss_elimination(A, b)

print(L2)
```

```
[[ 1.  1.  1.]
 [ 7.  4. -1.]
 [ 2. -2.  1.]] [1. 2. 3.]
[[ 0.65517241]
 [-0.44827586]
 [ 0.79310345]]
```

In [ ]:

```python
def ThomasDecomposition(d,a = 0,b = 0,c = 0,A_in = [],isMatrix = False):

    n = len(d)    # order of tridiagonal square matrix

    # use a,b,c to create matrix A, which is not necessary in the algorithm
    if not isMatrix:
        A = np.zeros((n,n)).astype(float)

        for i in range(n):
            A[i,i] = b
            if i > 0:
                A[i, i-1] = a
            if i < n-1:
                A[i, i+1] = c
    else:
        A = A_in.copy()

    # new list of modified coefficients
    c_1 = [0]*n
    d_1 = [0]*n

    for i in range(n):
        if not i:
            c_1[i] = c/b
            d_1[i] = d[i] / b
        else:
            c_1[i] = c/(b-c_1[i-1]*a)
            d_1[i] = (d[i]-d_1[i-1]*a)/(b-c_1[i-1] * a)

    # x: solution of Ax=d
    x = [0]*n

    for i in range(n-1, -1, -1):
        if i == n-1:
            x[i] = d_1[i]
        else:
            x[i] = d_1[i]-c_1[i]*x[i+1]

    x = [round(_, 4) for _ in x]

    return x
```

ex1:\ 对下列5对角阵进行LU分解,并计算该过程需要的乘除法次数

$$A = \begin{pmatrix} 35 & -16 & 1 & 0 & 0 & \cdots & 0 \\ -16 & 35 & -16 & 1 & 0 & \cdots & 0 \\ 1 & -16 & 35 & -16 & 1 & \cdots & 0 \\ 0 & 1 & -16 & 35 & -16 & \cdots & 0 \\ 0 & 0 & 1 & -16 & 35 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & 35 \end{pmatrix}_{n*n}$$

In [ ]:

```python
n = 5 ##假设为5阶矩阵

A = np.zeros((n,n))
for i in range(n):
    for j in range(n):
        if abs(i-j) == 1:
            A[i,j] = -16
        elif abs(i-j) == 2:
            A[i,j] = 1
        elif i == j:
            A[i,j] = 35
        else:
            A[i,j] = 0

print(A)

L,U,_ = L_U_decomposition(A)

print("LU分解结果为:\nL\n",L,"\nU\n",U)
```

```
[[ 35. -16.   1.   0.   0.]
 [-16.  35. -16.   1.   0.]
 [  1. -16.  35. -16.   1.]
 [  0.   1. -16.  35. -16.]
 [  0.   0.   1. -16.  35.]]
LU分解结果为:
L
 [[ 1.          0.          0.          0.          0.        ]
 [-0.45714286  1.          0.          0.          0.        ]
 [ 0.02857143 -0.56140351  1.          0.          0.        ]
 [ 0.          0.03611971 -0.58823529  1.          0.        ]
 [ 0.          0.          0.0381016  -0.59545455  1.        ]]
U
 [[ 35.         -16.           1.           0.           0.        ]
 [  0.          27.68571429 -15.54285714   1.           0.        ]
 [  0.           0.          26.24561404 -15.43859649   1.        ]
 [  0.           0.           0.          25.88235294 -15.41176471]
 [  0.           0.           0.           0.          25.78489305]]
```

ex2:\ 利用追赶法计算三对角矩阵$Ax = b$,$n = 15$;并画出向量x的图像 $$ A =

$$\begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & & & \\ & & \ddots & & \\ & & & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}$$

_{n*n}

x =

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix}$$

_{n*1}

b =

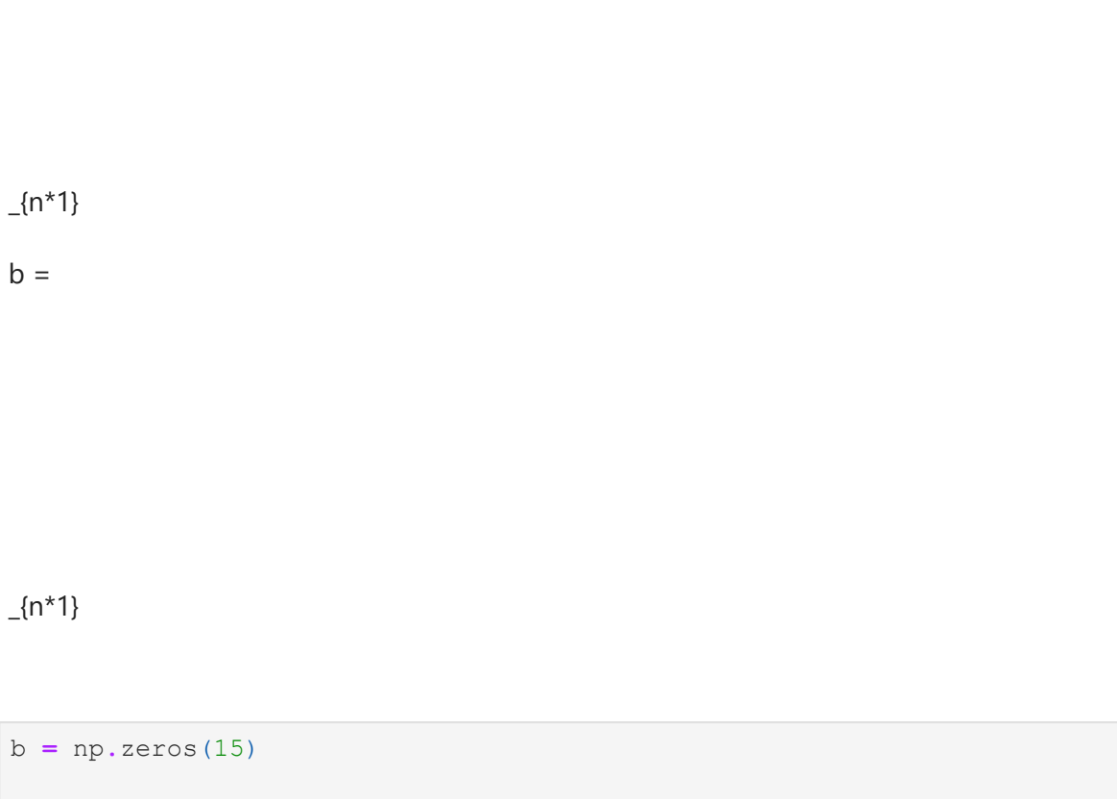
$$\begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}$$

_{n*1}

In [ ]:

```python
b = np.zeros(15)

print(len(b))
b[0] = 1
print(b)

x = ThomasDecomposition(b,-1,2,-1)

index = np.arange(15)

plt.plot(index,x)

print(x)
```

```
15
[1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0.9375, 0.875, 0.8125, 0.75, 0.6875, 0.625, 0.5625, 0.5, 0.4375, 0.375, 0.3125, 0.25, 0.1875, 0.125, 0.0625]
```



ex3;对矩阵A进行LU分解以及QR分解;并对$A^T A$进行Cholesky分解;

$$A = \begin{pmatrix} 8 & -3 & 2 \\ 4 & 11 & -1 \\ 6 & 3 & 12 \end{pmatrix}$$

In [ ]:

```python
A = np.array([[8,-3,2], [4, 11, -1], [6, 3, 12]]).astype(float)

B = A.T@A
print("A生成的正定矩阵为:\n",B)

L,U,_ = L_U_decomposition(A)

Q,R = Q_R_decomposition(A)

LCho,D = Cholesky_decomposition(B)

print("其LU分解为:\nL\n",L,"\nU\n",U,"\n")
print("其QR分解为:\nQ\n",Q,"\nR\n",R,"\n","\nQ与R乘积为:\n",np.dot(Q,R))

print("其cholesky分解为:\nL\n",LCho,"\nD\n",D,"\n")
```

```
A生成的正定矩阵为:
 [[116.  38.  84.]
 [ 38. 139.  19.]
 [ 84.  19. 149.]]
其LU分解为:
L
 [[1.   0.   0.  ]
 [0.5  1.   0.  ]
 [0.75 0.42 1.  ]]
U
 [[ 8.   -3.    2.  ]
 [ 0.   12.5  -2.  ]
 [ 0.    0.   11.34]]

其QR分解为:
Q
 [[ 0.74278135 -0.49963813 -0.44568779]
 [ 0.37139068  0.86133935 -0.34664606]
 [ 0.55708601  0.09195794  0.82534775]]
R
 [[ 1.07703296e+01  3.52821143e+00  7.79920420e+00]
 [ 8.32667268e-17  1.12495211e+01 -7.57120355e-01]
 [-1.11022302e-15 -1.11022302e-16  9.35944350e+00]]

Q与R乘积为:
 [[ 8. -3.  2.]
 [ 4. 11. -1.]
 [ 6.  3. 12.]]
其cholesky分解为:
L
 [[ 1.          0.          0.        ]
 [ 0.32758621  1.          0.        ]
 [ 0.72413793 -0.06730245  1.        ]]
D
 [[116.           0.           0.        ]
 [  0.         126.55172414   0.        ]
 [  0.           0.          87.59918256]]
```