In [ ]:

```python
import numpy as np

import matplotlib.pyplot as plt

import scipy as sp

import sympy as sym
```

对于求解方程组而言,我们的讨论已经足以解决一大部分数值解的问题,而当我们将目光放到系数矩阵的计算时,发现问题变得比较复杂.\ `Definition` 特征值与特征向量:\ 设$A \in \R^{n\times n}$,$\lambda \in C$和非零向量$x \in \R^{n}$满足$Ax = \lambda x$,则λ称为A的特征值,而x称为A的特征向量.\ 因此,求解特征值就意味着求解等价问题:

$$(A - \lambda I)x = 0$$

在结束了特征值和特征向量定义后,我们再来补充一些特征值的性质: `Method1` 线性运算性: 设λ为$A \in \R^{n\times n}$的特征值,则有:

1. c λ 为c A 的特征值
2. $\lambda - \mu$为$(A - \mu I)$的特征值
3. λ^k为A^k的特征值

`Method2` 充要条件:

1. 矩阵A可相似对角化的充要条件是A有n个线性无关的特征向量.
2. 矩阵A的不同特征值构成的特征子空间正交,即不同特征值对应的特征向量正交.

`Method3`

In [ ]:

```python
def Rayleigh(A,x):
    """
    瑞利商定义

    Args:
        A (Matrix): 要求对称的系数矩阵
        x (vector): 特征向量

    Returns:
        float:瑞利商的值
    """

    b = np.dot(A,x)

    return np.dot(x,b)/np.dot(x,x)
```

In [ ]:

```python
A = np.array([
    [2,5,0],
    [5,3,7],
    [0,7,6]
])

lam,x = np.linalg.eig(A)

print(Rayleigh(A,x[2]))
```

```
-3.7754908865633916
```

In [ ]:

```python
def PowerMethod(A,v0 = None,isRayleigh = False,tol = 1e-6,max_iter = 150,output_key = False):
    """幂法求解矩阵特征值

    Args:
        A (Matrix): 系数矩阵
        v0 (vector): 迭代初始变量
        Rayleigh (bool, optional): 采用瑞利商计算最大值. Defaults to False.
        tol (float, optional): 精度限. Defaults to 1e-6.
        max_iter (int, optional): 最大迭代上限. Defaults to 150.
        putout_key (bool, optional): 输出迭代信息. Defaults to False.
    """

    A = A.copy()

    if v0 is None:
        v0 = np.random.rand(A.shape[0])
    else:
        v0 = v0.copy()


    u0 = v0

    oldmu = max(v0)

    i = 0

    while i < max_iter:

        v0 = A @ u0

        mu = max(v0)

        u0 = v0 / mu

        if(abs(mu - oldmu)<tol):
            break

        oldmu = mu

        i += 1

    if i < max_iter and output_key:

        if isRayleigh:
            mu = Rayleigh(A,u0)

        print(f"迭代{i+1}次收敛,特征值为:{mu},特征向量为:{u0}")

    elif i == max_iter and output_key:
        print(f"迭代{i+1}次未收敛,特征值为:{mu},特征向量为:{u0}")


    return mu,u0
```

In [ ]:

```python
A = np.array([
    [1.0,1.0,0.5],
    [1.0,1.0,0.25],
    [0.5,0.25,2.0]
])

lam,vec = PowerMethod(A,output_key=True)

## 原点平移法加速

p = 0.75

B = A - p*np.eye(A.shape[0])

lam1,vec1 = PowerMethod(B,output_key=True)

## 瑞利商加速

lam2,vec2 = PowerMethod(A,isRayleigh=True,output_key=True)
```

```
迭代19次收敛,特征值为:2.536527100235017,特征向量为:[0.7482221  0.64966214 1.        ]
迭代17次收敛,特征值为:1.7865262714363126,特征向量为:[0.74822143 0.64966136 1.        ]
迭代26次收敛,特征值为:2.5365258604166776,特征向量为:[0.74822209 0.64966213 1.        ]
```

下面我们来介绍Householder变换，以及Householder变换的矩阵特征值计算\ Definition:\ 设向量$\omega \in \R^{n}$,且其满足$\omega^T\omega = 1$,即向量2-范数为1,称矩阵

$$H(\omega) = I - 2\omega\omega^T$$

为初等反射矩阵,称为Householder变换.如果设$\omega = (\omega_1, \cdots, \omega_n)^T$,则:

$$H = \begin{bmatrix} 1-2\omega_1^2 & -2\omega_1\omega_2 & \cdots & -2\omega_1\omega_n \\ -2\omega_2\omega_1 & 1-2\omega_2^2 & \cdots & -2\omega_2\omega_n \\ \vdots & \vdots & \ddots & \vdots \\ -2\omega_n\omega_1 & -2\omega_n\omega_2 & \cdots & 1-2\omega_n^2 \end{bmatrix}$$

In [ ]:

```python
def householder(omega):
    """Householder transformations

    Args:
        omega (vector): 向量
    """

    n = omega.shape[0]

    x = omega.copy()

    sigma = np.sign(x[0]) * np.linalg.norm(x)

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

    e[0] = 1

    u = x + sigma * e

    beta = np.linalg.norm(u) ** 2 / 2  ##要求系数矩阵非奇异

    I = np.eye(n)

    H = I - u@u.T / beta

    return H


def householder_tran(x):
    x = x.copy()

    x = x / np.linalg.norm(x)

    n = x.shape[0]

    i = np.eye(n)

    H = i - 2 * x @ x.T

    return H
```

In [ ]:

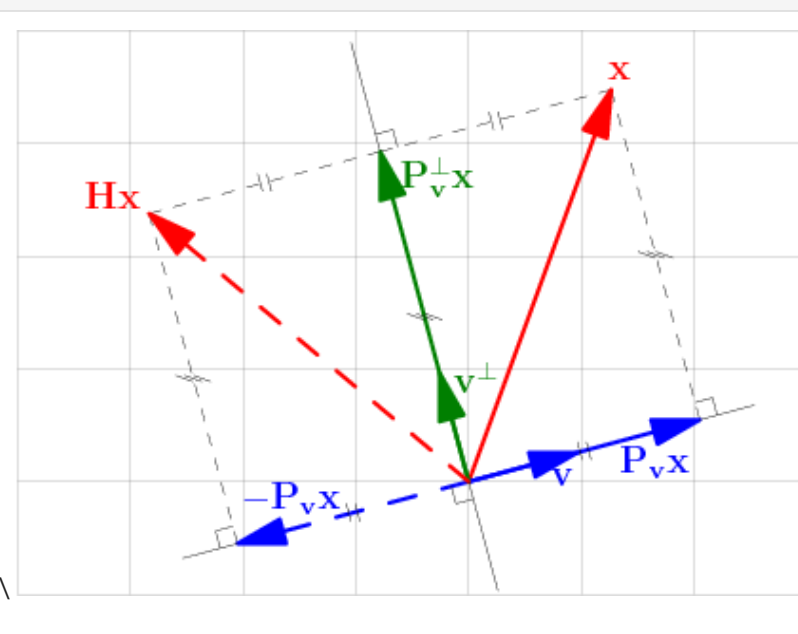
```python
x = np.array([
    [1],
    [0]
])

y = np.array([
    [0.6],
    [0.8]
])

H = householder_tran(x)

y_t = H @ y

x,y,y_t = x.reshape(-1),y.reshape(-1),y_t.reshape(-1)
```



Householder变换存在的目的即为将一个向量按照另一向量的垂直方向做镜像对应.\ 这种操作可以将任一非零向量转换为空间内单位向量\ 即:\ 设$x = (x_1, \cdots, x_n)^T \neq 0$,则存在初等反射矩阵$H$使$Hx = -\sigma e_i$,其中

$$\begin{cases} H = I - \beta^{-1}uu^T \\ \sigma = sgn(x_i)\|x\|_2 \\ u = x + \sigma e_i \\ \beta = \dfrac{\|u\|_2^2}{2} \end{cases}$$

In [ ]:

```python
def QR_householder_decomposition(A,output_key=False):
    """采用Householder变换进行QR分解

    Args:
        A (matrix): 系数矩阵
        output_key (bool, optional): 输出控制. Defaults to False.
    """

    A = A.copy()

    n = A.shape[0]

    Q = np.eye(n)

    i = 0

    for x in A.T:

        y = x[i:].reshape(-1,1)

        H_i = householder(y)

        A[i:,i:] = H_i @ A[i:,i:]


        I_i = np.eye(i)

        H = np.block([
            [I_i,np.zeros((i,n-i))],
            [np.zeros((n-i,i)),H_i]
        ])

        Q = Q @ H

        i+=1


    Q = np.around(Q,decimals=8)
    R = np.around(A,decimals=8)

    if output_key:
        print(f"经过{i+1}次householder变换后,QR分解结果为:\nQ=\n{Q}\nR=\n{R},\nQR乘积为:\n{np.around(Q@R,decimals=4)}")

    return Q,R
```

In [ ]:

```python
A = np.array([[12, -51, 4],
              [6, 167, -68],
              [-4, 24, -41]]).astype(float)

Q,R = QR_householder_decomposition(A,output_key = True)
```

```
经过4次householder变换后,QR分解结果为:
Q=
[[-0.85714286  0.39428571 -0.33142857]
 [-0.42857143 -0.90285714  0.03428571]
 [ 0.28571429 -0.17142857 -0.94285714]]
R=
[[ -14.  -21.   14.]
 [   0. -175.   70.]
 [  -0.   -0.   35.]],
QR乘积为:
[[ 12. -51.   4.]
 [  6. 167. -68.]
 [ -4.  24. -41.]]
```

在介绍完镜面反射后,我们来介绍旋转反射,即Givens旋转.\ Givens旋转是旋转矩阵的另一种形式,它只对矩阵的列进行旋转,而不对行进行旋转,同时保持矩阵的行列式不变且仍具有正交性.

In [ ]:

```python
def givens_trans(n,i,j,theta):
    """生成Givens变换矩阵

    Args:
        n (_type_): _description_
        i (_type_): _description_
        j (_type_): _description_
        theta (_type_, optional): _description_.
    """

    G = np.eye(n)

    G[i,i] = np.cos(theta)
    G[i,j] = np.sin(theta)
    G[j,i] = -np.sin(theta)
    G[j,j] = np.cos(theta)

    return G
```

In [ ]:

```python
def givens(x):
    """givens约化

    Args:
        x (_type_): _description_
    """

    x = x.copy()

    n = x.shape[0]

    P = np.eye(n)

    i = 0

    for j in range(1,n):

        theta = np.arctan(x[j] / x[i])

        P_i = givens_trans(n,i,j,theta)

        x = P_i @ x

        P = P_i @ P

    return P
```

In [ ]:

```python
x = np.array([
    [0.6]
]).astype(np.float32)

x = np.reshape(x,(-1,1))

P = givens(x)
```

In [ ]:

```python
def QR_givens_decomposition(A,output_key = False):
    """采用吉文斯变换的 QR 分解

    Args:
        A (_type_): _description_
        ouput_key (_type_): _description_
    """

    A = A.copy()

    n = A.shape[0]

    Q = np.eye(n)

    i = 0

    for j in range(n):
        x = A.T[j]

        y = np.reshape(x[i:],(-1,1))

        P_i = givens(y)

        I_i = np.eye(i)

        P = np.block([
            [I_i,np.zeros((i,n-i))],
            [np.zeros((n-i,i)),P_i]
        ])

        A = P @ A

        Q = P @ Q

        i += 1

    R = np.around(A,decimals=5)
    Q = np.around(Q.T,decimals=5)

    if output_key:
        print(f"矩阵经{i+1}次Givens列变换后,得到的分解为:\nQ:\n{Q}\nR:\n{R}\nQR乘积为:\n{np.around(Q@R,decimals = 0)}")

    return Q,R
```

In [ ]:

```python
A = np.array([[12, -51, 4],
              [6, 167, -68],
              [-4, 24, -41]]).astype(np.float32)

Q,R = QR_givens_decomposition(A,output_key = True)
```

```
矩阵经4次Givens列变换后,得到的分解为:
Q:
[[ 0.85714 -0.39429  0.33143]
 [ 0.42857  0.90286 -0.03429]
 [-0.28571  0.17143  0.94286]]
R:
[[ 14.  21. -14.]
 [ -0. 175. -70.]
 [ -0.   0. -35.]]
QR乘积为:
[[ 12. -51.   4.]
 [  6. 167. -68.]
 [ -4.  24. -41.]]
```

上述分析可知,通过Givens变换或者householder变换可以将一个矩阵变换为正交矩阵乘子形式.下面我们来介绍在计算特征值上更为实用的QR方法.

`Householder QR`:

1. 通过householder变换将矩阵变换为上Hessenberg矩阵.
2. 通过不同的QR方法计算特征值与特征向量.

In [ ]:

```python
def householder_trans_hessenberg(A,output_key = False):
    """_summary_

    Args:
        A (_type_): _description_
        output_key (bool, optional): _description_. Defaults to False.
    """

    A = A.copy()

    n = A.shape[0]

    i = 1

    U = np.eye(n)

    for x in A.T:

        c = x[i:].reshape(-1,1)

        sigma = np.sign(c[0]) * np.linalg.norm(c)

        e = np.zeros((n-i,1))
        e[0] = 1

        u = c + sigma * e

        beta = sigma*(sigma + c[0])

        I_i = np.eye(i)

        R_i = np.eye(n-i) - 1/beta * u@u.T

        U_i = np.block([
            [I_i,np.zeros((i,n-i))],
            [np.zeros((n-i,i)),R_i]
        ])

        A = U_i @ A @ U_i

        U = U_i @ U

        i+=1

        if i == n-1:

            break

    A = np.around(A,decimals = 6)
    U = np.around(U,decimals = 6)

    if output_key:

        print(f"经过{i+1}次迭代,由householder变换生成的矩阵为:\n{A}")

    return A,U
```

In [ ]:

```python
A = np.array([
    [-4,-3,-7],
    [2,3,2],
    [4,2,7]
]).astype(np.float64)

H,U = householder_trans_hessenberg(A,True)
```

```
经过3次迭代,由householder变换生成的矩阵为:
[[-4.        7.602631 -0.447214]
 [-4.472136  7.8      -0.4     ]
 [ 0.       -0.4       2.2     ]]
```

当我们将矩阵约化为上Hessenberg矩阵后,其特征值我们将有一套很好的算法即QR方法来处理这一类问题.

In [ ]:

```python
def QR_method(A,isSymmetric = False,output_key = False,tol = 1e-5,max_iter = 50):
    """
    QR方法求解矩阵特征值

    Args:
        A (_type_): _description_
        output_key (bool, optional): _description_. Defaults to False.
        tol (_type_, optional): _description_. Defaults to 1e-6.
    """

    A = A.copy()

    if np.allclose(A,A.T):
        isSymmetric = True

    n = A.shape[0]

    if isSymmetric:

        U = np.eye(n)

    A,_ = householder_trans_hessenberg(A)

    ##print(A)

    def sumOfUpper(A):

        res = 0

        for i in range(A.shape[0]):
            for j in range(0,i):
                res += A[i][j] ** 2

        return res ** 0.5

    def diag(A):
        res = []

        for i in range(A.shape[0]):
            res.append(A[i][i])

        return res

    i = 0

    while i < max_iter:

        s_i = A[-1][-1]

        I_i = np.eye(n)

        Q,R = QR_householder_decomposition(A)

        if isSymmetric:
            U = U @ Q

        A = R @ Q

        i += 1

        if sumOfUpper(A) < tol:
            break

    if i == max_iter:
        print("迭代次数过多, 无法收敛")
        return None
    else:

        if output_key:
            print(f"矩阵经分解后特征值为\n{A}\n,特征向量为:\n{U}")

        if isSymmetric:

            eigvector = np.around(U,decimals = 5)
            eigvalue = np.around(A,decimals = 5)

            return eigvalue,eigvector

        else:

            eigvector = []
            eigvalue = np.around(A,decimals = 5)

            return eigvalue,eigvector
```

In [ ]:

```python
A = np.array([
    [2,1,0],
    [1,3,1],
    [0,1,4]
]).astype(float)

eigvalue,eigvector = QR_method(A,output_key=True)
```

```
矩阵经分解后特征值为
[[ 4.73205086e+00 -8.68901403e-06  1.00000000e-07]
 [-8.60999986e-06  2.99999995e+00 -8.00000000e-08]
 [ 0.00000000e+00  0.00000000e+00  1.26794920e+00]]
,特征向量为:
[[-0.21132777 -0.57734921 -0.78867512]
 [ 0.57735317  0.57734736 -0.57735027]
 [-0.78867225  0.5773542  -0.2113249 ]]
```

上机练习1:\ 利用householder变换约化矩阵:

$$A = \begin{pmatrix} 1 & 3 & 4 \\ 3 & 1 & 5 \\ 4 & 2 & 1 \end{pmatrix}$$

为上Hessenberg矩阵H,求出反射矩阵P使得 $H = PAP$

In [ ]:

```python
A = np.array([
    [1,3,4],
    [3,1,5],
    [4,2,1]
]).astype(np.float64)

H,P = householder_trans_hessenberg(A)

print(f"经householder约化后矩阵\n{A}\n变换为:\n{H},\n其反射矩阵\n{P}\n")
```

```
经householder约化后矩阵
[[1. 3. 4.]
 [3. 1. 5.]
 [4. 2. 1.]]
变换为:
[[ 1.   -5.   -0.  ]
 [-5.    4.36 -0.52]
 [-0.    2.48 -2.36]],
其反射矩阵
[[ 1.   0.   0. ]
 [ 0.  -0.6 -0.8]
 [ 0.  -0.8  0.6]]
```

上机练习2:用幂法、反幂法计算矩阵最大最小特征值对,以及QR算法计算

$$A = \begin{pmatrix} 1 & 1 & 1/2 \\ 1 & 1 & 1/4 \\ 1/2 & 1/4 & 2 \end{pmatrix}$$

的所有特征值和特征向量

In [ ]:

```python
A = np.array([
    [1, 1, 1/2],
    [1, 1, 1/4],
    [1/2, 1/4, 2]
]).astype(np.float64)

lambda1 = PowerMethod(A)

print(f"由幂法得到的特征值:\n{lambda1}\n")

eigvalue,eigvector = QR_method(A)

print(f"由QR方法得到的特征值:\n{eigvalue}\n,特征向量\n{eigvector}\n")
```

```
由幂法得到的特征值:
(2.5365245826319622, array([0.74822017, 0.64966012, 1.        ]))

由QR方法得到的特征值:
[[ 2.53653e+00 -1.00000e-05  0.00000e+00]
 [-1.00000e-05  1.48012e+00  0.00000e+00]
 [ 0.00000e+00  0.00000e+00 -1.66500e-02]]
,特征向量
[[ 0.53149  0.44428  0.72121]
 [-0.73042 -0.19078  0.65581]
 [ 0.42896 -0.87534  0.22311]]
```

书面作业2:\ 用幂法计算下列矩阵的主特征值和特征向量

$$A = \begin{pmatrix} 7 & 3 & -2 \\ 3 & 4 & -1 \\ -2 & -1 & 3 \end{pmatrix}$$

In [ ]:

```python
A =np.array([
    [7,3,-2],[3,4,-1],[-2,-1,3]
]).astype(np.float64)

lamb = PowerMethod(A)

print(lamb)
```

```
(9.605551150657337, array([ 1.        ,  0.60555138, -0.39444857]))
```