```python
import numpy as np

import matplotlib.pyplot as plt

import sympy as sp

import nbconvert

import pandoc

from IPython.display import display, Math, Latex
```

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

norm1 = np.linalg.norm(A,ord = 1)
norm2 = np.linalg.norm(A,ord = 2)
norm_inf = np.linalg.norm(A,ord = np.inf)

print(norm1,norm2,norm_inf)
```

```
6.0 5.464985704219043 7.0
```

```python
def tril(A,index = 0):
    """
    得到A矩阵第index行以下的矩阵

    Args:
        A (_Matrix_): 输入矩阵
    """
    n=len(A)

    L = np.zeros_like(A)

    for i in range(index,n):
        for j in range(0,i+1-index):
            L[i,j] = A[i,j]

    return L


def diag(A):
    D = np.zeros_like(A)

    for i in range(len(A)):
        D[i,i] = A[i,i]

    return D


def triu(A,index = 0):
    """
    得到A矩阵第index行以上的矩阵

    Args:
        A (_Matrix_): 输入矩阵
    """
    n=len(A)

    U = np.zeros_like(A)

    for j in range(index,n):
        for i in range(0,j+1-index):
            U[i,j] = A[i,j]

    return U


def Jacobi_interation(A,b,x0,tol = 1e-6,max_iter = 50):
    """
    采用Jacobi迭代法求解线性方程组Ax=b

    Args:
        A (_type_): _description_
        b (_type_): _description_
        x0 (_type_): _description_
        tol (_type_): _description_
        max_iter (_type_): _description_
    """

    dA = diag(A)
    r0 = np.linalg.norm(b - np.dot(A,x0))
    L = tril(A,1)
    U = triu(A,1)
    B = L + U

    for k in range(max_iter):
        x1 = np.dot(np.linalg.inv(dA),b - np.dot(B,x0))
        r1 = np.linalg.norm(b - np.dot(A,x1))
        if r1/r0 < tol:
            print(f"迭代次数:{k+1},结果为:{x1}")
            break
        else:
            x0 = x1

    if k == max_iter-1:
        print("迭代次数已达到最大值")

    return x1
```

```python
def Gauss_Seidel_interation(A,b,x0,tol = 1e-6,max_iter = 50):
    """采用Gauss_seidel法求解线性方程组Ax=b

    Args:
        A (_type_): _description_
        b (_type_): _description_
        x0 (_type_): _description_
        tol (_type_): _description_
        max_iters (_type_): _description_
    """

    dA = diag(A)
    r0 = np.linalg.norm(b - np.dot(A,x0))
    L = tril(A,1)
    U = triu(A,1)
    B = L + U

    for k in range(max_iter):
        x1 = np.dot(np.linalg.inv(dA - L),b + np.dot(U,x0))
        r1 = np.linalg.norm(b - np.dot(A,x1))
        if r1/r0 < tol:
            print(f"迭代次数:{k+1},结果为:{x1}")
            break
        else:
            x0 = x1

    if k == max_iter-1:
        print("迭代次数已达到最大值")

    return x1
```

```python
def Successive_over_relaxation(A,b,x0,w = 0.15,tol = 1e-6,max_iter = 50):
    """采用超松弛迭代方法求解线性方程组

    Args:
        A (_type_): _description_
        b (_type_): _description_
        w (_type_): _description_
        tol (_type_): _description_
        max_iter (_type_): _description_

    Returns:
        _type_: _description_
    """

    dA = diag(A)
    r0 = np.linalg.norm(b - np.dot(A,x0))
    L = tril(A,1)
    U = A - L


    for k in range(max_iter):
        C = np.linalg.inv(dA - w*L)
        x1 = np.dot(np.dot(np.linalg.inv(dA - w * L),((1-w) * dA + w * U)),x0) + w * np.dot(np.linalg.inv(dA - w*L),b)
        r1 = np.linalg.norm(b - np.dot(A,x1))
        if r1/r0 < tol:
            print(f"迭代次数:{k+1},结果为:{x1}")
            break
        else:
            x0 = x1 + w*(x1 - x0)

    if k == max_iter-1:
        print("迭代次数已达到最大值")

    return x1
```

```python
def grad_descent(A,b,x0,tol = 1e-6,max_iter = 150):
    """
    采用梯度下降法计算Ax = b的解

    Args:
        A (_Matrix_): 系数矩阵
        b (_Vector_): 偏置
        x0 (_Vector_): 初始向量
    """
    A = A.copy()
    b = b.copy()
    x0 = x0.copy()

    r0 = b - np.dot(A,x0)   ## 初始残差

    i = 0 ## 迭代次数

    while np.linalg.norm(r0) > tol and i < max_iter:

        P = r0

        A_P = np.dot(A,P)

        alpha = np.dot(P,r0)/np.dot(P,A_P)

        x0 = x0 + alpha * P

        r0 -= alpha * A_P

        i += 1


    if i < max_iter:
        x = x0

        print(f"Iterator:{i+1}时收敛到目标精度,求解出的解为:{x}")


    else:
        residual = np.linalg.norm(r0)
        x = x0

        print(f"Iterator:{i+1},但是未达到目标精度, 残差为{residual}")


    return x
```

```python
A = np.array(
    [
        [10,0],
        [0,1]
    ]
).astype(np.float64)

b = np.array([8,5]).astype(np.float64)

x0 = np.array([1,1]).astype(np.float64)

x = grad_descent(A,b,x0)
print(x)
```

```
Iterator:55时收敛到目标精度,求解出的解为:[0.80000004 4.99999921]
[0.80000004 4.99999921]
```

我们很容易能够发现,梯度下降法每次前进后,下一次的方向与上一次的方向必定正交,即:

$$P_{k+1} \perp P_k$$

这就意味着,我们在每次前进后必须沿着垂直的方向前进,尽管我们在当前时间步内取得了局部的最小值,但这并不意味着在全局内我们是最优策略,转化为Code方面的来看,也就是grad_descent采用的是 `greedy` (贪心)策略,但是实际上,我们面临的问题并不都是贪心的.\ 在这种意义上,我们或许需要寻找一个相对来说收敛更快的的算法\ 这也就引出了我们下面定义的共轭梯度法

```python
def AspaceDot(A,u,v):
    """
    A-内积定义

    Args:
        A (_Matrix_): A矩阵(要求对称正定)
        u (_Vector_): 内积向量
        v (_Vector_): 内积向量

    Returns:
        _float/integer_: 返回u,v在A矩阵意义下的内积
    """

    return np.dot(u,A@v)


def conjugate_gradient(A,b,x0,tol=1e-6,max_iter=150):
    """采用共轭梯度法来求解Ax=b的解

    Args:
        A (_type_): _description_
        b (_type_): _description_
        x0 (_type_): _description_
        tol (_type_, optional): _description_. Defaults to 1e-6.
        maxiter (int, optional): _description_. Defaults to 150.
    """

    A = A.copy()
    b = b.copy()
    x0 = x0.copy()


    r = b - A@x0
    P = list()
    P.append(r)

    i = 0

    while i < max_iter and np.linalg.norm(r)>tol:

        alpha = np.dot(P[-1],r) / np.dot(P[-1],A@P[-1])

        x0 = x0 + alpha*P[-1]
        r = b - A@x0

        t = r

        length = len(P)

        for j in range(length):
            t = t - AspaceDot(A,P[j],r) / AspaceDot(A,P[j],P[j]) * P[j]

        P.append(t)

        i += 1

    if i == max_iter:

        x = x0

        residual = np.linalg.norm(r)

        print(f"Iterator:{max_iter},算法未收敛,残差为:{residual},x为:{x0}")

    else:

        x = x0

        print(f"Iterator:{i},算法收敛,x为:{x}")

    return x
```

```python
A = np.array(
    [
        [10,0],
        [0,1]
    ]
).astype(np.float64)

b = np.array([8,5]).astype(np.float64)

x0 = np.array([1,1]).astype(np.float64)

x = conjugate_gradient(A,b,x0)
print(x)
```

```
Iterator:2,算法收敛,x为:[0.8 5. ]
[0.8 5. ]
```

上机练习1:\ 分别利用 `Jacobi,Gauss-Seidel,SOR迭代法` 计算 `Ax = b` , `n = 15` 并画出向量 `x` 的图像 $$ A =

$$\begin{pmatrix} \frac{5}{2} & -1 & 0 & \cdots & 0 & -1 \\ -1 & \frac{5}{2} & -1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots & \\ -1 & 0 & \cdots & 0 & -1 & \frac{5}{2} \end{pmatrix}$$

_{n\times n}

x =

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

_{n\times 1}

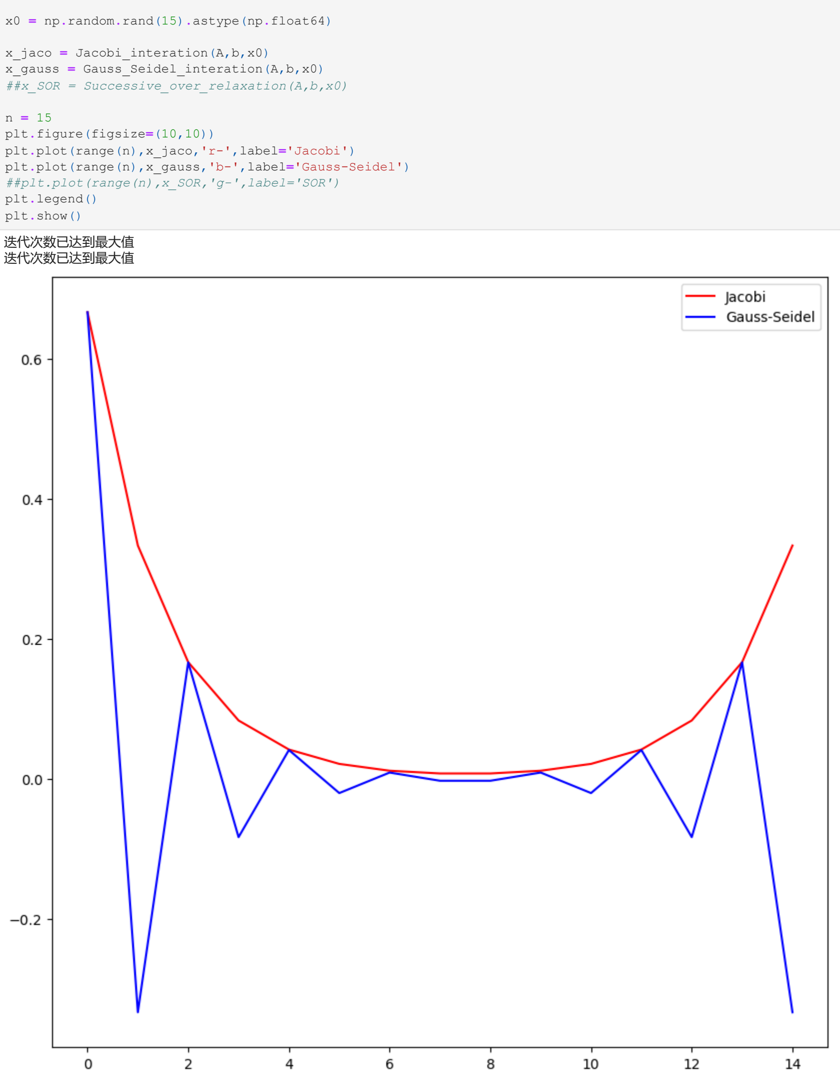
b =

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

_{n\times 1} $$

```python
A = np.zeros((15,15)).astype(np.float64)

for i in range(15):
    A[i,i] = 5/2
    if i != 0:
        A[i,i-1] = -1
    if i != 14:
        A[i,i+1] = -1
A[0,14],A[14,0] = -1,-1

b = np.zeros(15).astype(np.float64)
b[0] = 1

x0 = np.random.rand(15).astype(np.float64)

x_jaco = Jacobi_interation(A,b,x0)
x_gauss = Gauss_Seidel_interation(A,b,x0)
##x_SOR = Successive_over_relaxation(A,b,x0)

n = 15
plt.figure(figsize=(10,10))
plt.plot(range(n),x_jaco,'r-',label='Jacobi')
plt.plot(range(n),x_gauss,'b-',label='Gauss-Seidel')
##plt.plot(range(n),x_SOR,'g-',label='SOR')
plt.legend()
plt.show()
```

```
迭代次数已达到最大值
迭代次数已达到最大值
```



ex2:\ 尝试自己写 `CG` 和 `Germs` 算法程序计算上述例子,体会计算效率的不同

```python
x = conjugate_gradient(A, b, x0, tol=1e-6, max_iter=150)

plt.plot(range(len(x)), x)
```

```
Iterator:8,算法收敛,x为:[0.66670736 0.3333842  0.16675314 0.08349864 0.04199347 0.02148503
 0.01171911 0.00781274 0.00781274 0.01171911 0.02148503 0.04199347
 0.08349864 0.16675314 0.3333842 ]
```

```
[<matplotlib.lines.Line2D at 0x17f3e8cbd00>]
```