IHBACH Mohamed Yassine

Numerical Linear Algebra

MSD1 , AL KHWARIZMI , UM6P

Les algorithmes suivants sont testés par l'exemple suivant :

$$\begin{bmatrix} 5 & 1 & 2 \\ 1 & 7 & 3 \\ 2 & 3 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 5 \\ 6 \end{bmatrix}$$ **qui a pour solution exacte** $$x = \begin{bmatrix} \frac{-3}{13} \\ \frac{4}{11} \\ \frac{128}{143} \end{bmatrix} = \begin{bmatrix} -0.230769 \\ 0.363636 \\ 0.895104 \end{bmatrix}$$

Algorithm 1 : GMRS

In [13]:
```
import numpy as np

def GMRES(A,b,x0,m,tol):

    while(1):
        r = b - A @ x0
        H = np.zeros([m,m+1])
        V = np.zeros([m,b.size])
        beta = np.linalg.norm(r)
        if( beta < tol ) : break
        e1 = np.zeros(m+1)
        e1[0] = 1
        V[0,:] = r / beta
        for j in range(m):
            w = A @ V[j,:]
            for i in range(j):
                H[j,i] = w @ V[i,:]
                w = w-H[j,i]*V[i,:]
            alpha = np.linalg.norm(w,ord=2)
            H[j,j+1]=alpha
            if (j==m-1) or (alpha==0):
                break;
            V[j+1,:]=w/alpha
        y= np.linalg.pinv(H.T) @ (beta * e1.T)
        x0= x0 + V.T @ y

    return x0


x0 , b , A  = np.zeros(3) , np.array([1,5,6]) , np.array([[5,1,2],[1,7,3
],[2,3,6]])
print("\nSolution X = ",GMRES(A,b,x0,3,0.001))
```

```
Solution X =  [-0.23061365  0.3637526   0.89484384]
```

Algorithm 2 : The symmetric Lanczos

In [16]:
```
import numpy as np

def Symmetric_Lanczos(A,b,x0,m):
    V = np.zeros([m,b.size])
    a = np.zeros(m)
    e1 = np.zeros(m);e1[0]=1
    bi = np.zeros(m)
    r = b - A @ x0
    betha = np.linalg.norm(r)
    v = r / betha
    V[0,:] = v
    for j in range(m):
        w = A @ V[j,:] -  bi[j]*V[j-1,:]
        a[j] = w @ V[j,:]
        w = w - a[j] * V[j,:]
        B = np.linalg.norm(w,ord=2)
        if (B==0) or (j==m-1):
              break
        bi[j+1] = B
        V[j+1,:] = w / B
    Tm = np.diag(a) + np.diag(bi[1:],1) + np.diag(bi[1:],-1)
    ym = betha * np.linalg.inv(Tm) @ e1
    V = V.T
    xm = x0 + V @ ym
    return xm

x0 , b , A  = np.zeros(3) , np.array([1,5,6]) , np.array([[5,1,2],[1,7,3
],[2,3,6]])
print("\nSolution X = ",Symmetric_Lanczos(A,b,x0,3))
```

```
Solution X =  [-0.23076923  0.36363636  0.8951049 ]
```

Algorithm 3 : The Nonsymmetric Lanczos

In [24]:
```
import numpy as np
def Nonsymmetric_Lanczos(A,b,x0,m):
    V = np.zeros([m,b.size])
    W = np.zeros([m,b.size])
    a = np.zeros(m)
    e1 = np.zeros(m);e1[0]=1
    bi = np.zeros(m)
    di = np.zeros(m)
    r = b - A @ x0
    betha = np.linalg.norm(r)
    v = r / betha
    V[0,:] = v
    W[0,:] = v
    for j in range(m):
        Av = A @ V[j,:]
        Atw = A.T @ W[j,:]
        a[j] = Av @ W[j,:]
        v = Av - a[j] * V[j,:] - bi[j] * V[j-1,:]
        w = Atw - a[j] * W[j,:] - di[j] * W[j-1,:]
        D = np.sqrt(v @ w.T)
        if (D==0) or (j==m-1):break;
        di[j+1]=D
        bi[j+1] = (v @ w.T) / D
        V[j+1,:] = v/bi[j+1]
        W[j+1,:] = w/di[j+1]
    Tm = np.diag(a) + np.diag(bi[1:],1) + np.diag(di[1:],-1)
    ym = betha * np.linalg.inv(Tm) @ e1
    V = V.T
    xm = x0 + V @ ym

    return xm

x0 , b , A  = np.zeros(3) , np.array([1,5,6]) , np.array([[5,1,2],[1,7,3
],[2,3,6]])
print("\nSolution X = ",Nonsymmetric_Lanczos(A,b,x0,5))
```

```
Solution X =  [-0.23076923  0.36363636  0.8951049 ]
```

Algorithm 4 : BICGSTAB

In [27]:
```
import numpy as np

def BICGSTAB(A,b,x0,jmax):
    r = b - A @ x0
    r0 = r.copy()
    p = r.copy()
    x = x0.copy()
    for j in range(jmax):
        Ap = A @ p
        a = (r.T @ r0) / (Ap.T @ r0)
        s = r-Ap*a
        As = A @ s
        w = (As.T @ s) / (As.T @ As)
        x = x+p*a+s*w
        rplus = s-As*w
        beta = (rplus.T @ r0) / (r.T @ r0)#*(a/w)
        p = rplus+(p-Ap*w)*beta
        r = rplus.copy()
    return x

x0 , b , A  = np.zeros(3) , np.array([1,5,6]) , np.array([[5,1,2],[1,7,3
],[2,3,6]])
print("\nSolution X = ",BICGSTAB(A,b,x0,3))
```

```
Solution X =  [-0.22053643  0.36882583  0.87040707]
```

Algorithm 5 : The BiConjugate Gradien

In [29]:
```
import numpy as np

def BCG(A,b,x0,jmax):
    r = b - A @ x0
    r0 = r.copy()
    r0 = r0/(np.linalg.norm(r0)**2)
    p = r.copy()
    p0 = r0.copy()
    x = x0.copy()
    for j in range(jmax):
        Ap = A.dot(p)
        a = (r.T @ r0)/ (Ap.transpose() @ p0 )
        x = x+p*a
        rplus = r-Ap*a
        Atp = A.T @ p0
        r0plus = r0 - Atp * a
        beta = (rplus.T @ r0plus )  /( r.T @ r0 ) #*(a/w)
        p = rplus+(p)*beta
        p0 = r0plus+(p0)*beta
        r = rplus.copy()
        r0 = r0plus.copy()
    return x


x0 , b , A  = np.zeros(3) , np.array([1,5,6]) , np.array([[5,1,2],[1,7,3
],[2,3,6]])
print("\nSolution X = ",BCG(A,b,x0,3))
```

```
Solution X =  [-0.23076923  0.36363636  0.8951049 ]
```

Algorithm 6 : The Golub_Kuhan Bidiagobalisation

In [34]:
```
import numpy as np

def GKB(A,u):
    [n,m] = A.shape
    U = np.zeros([m,n])
    V = np.zeros([m,n])
    a = np.zeros([n])
    b = np.zeros([n])
    U[0,:] = u
    a[0] = np.linalg.norm(A.T @  U[0,:])
    V[0,:] = (A.T @ U[0,:]) / a[0]
    for i  in range(1,m):
        Av = A @ V[i-1,:]
        U[i,:] = Av - a[i-1] * U[:,i-1]
        b[i] = np.linalg.norm(U[i,:])
        U[i,:] = U[i,:] / b[i]
        V[i,:] = (A.T @  U[i,:]) - b[i] * V[i-1,:]
        a[i] = np.linalg.norm(V[i,:])
        V[i,:] = V[:,i] / a[i]
    return U.T , V.T

u1 , A  = np.array([1,0,0]) , np.array([[5,1,2],[1,7,3],[2,3,6]])
u , v = GKB(A,u1)
print("========== Matrice A ========")
display(A)
print("========== Matrice U ========")
display(u)
print("========== Matrice V ========")
display(v)
```

```
========== Matrice A ========

array([[5, 1, 2],
       [1, 7, 3],
       [2, 3, 6]])

========== Matrice U ========

array([[1.        , 0.        , 0.34777364],
       [0.        , 0.58430473, 0.2064615 ],
       [0.        , 0.81153434, 0.91456391]])

========== Matrice V ========

array([[0.91287093, 0.02363654, 0.04423196],
       [0.18257419, 0.71176938, 0.        ],
       [0.36514837, 0.        , 0.82399313]])
```