# TP12021

December 14, 2021

## 1 TP 1

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

## 2 Exercice 1

question 1

[3]:
```
# On définit la matrice
A = np.array([[4,6,-2,3], [2,-1,0,1], [-7,0,1,12]],dtype= "float")
# on l'affiche sur la sortie
print("A= ", A)
```

```
A=  [[ 4.  6. -2.  3.]
 [ 2. -1.  0.  1.]
 [-7.  0.  1. 12.]]
```

[4]:
```
# on affiche une valeur
print(A[0,3])
```

```
3.0
```

question 2

[5]:
```
# on concatène verticalement ( vstack)
C = np.vstack((2*A[0:2,:],1/3*A[2,:]))
print("C= ",C)
```

```
C=  [[ 8.         12.         -4.          6.        ]
 [ 4.         -2.          0.          2.        ]
 [-2.33333333  0.          0.33333333  4.        ]]
```

question 3

[6]:
```
print(np.ones([1,3]))
B = np.vstack([np.arange(4,7,1),np.arange(5,16,5),np.ones([1,3])])
print("B = ", B)
```

```
[[1. 1. 1.]]
B =  [[ 4.  5.  6.]
```

```
 [ 5. 10. 15.]
 [ 1.  1.  1.]]
```

question 4

[7]:
```
C = A[0:3,0:3]
print("C = ", C)
```

```
C =  [[ 4.  6. -2.]
 [ 2. -1.  0.]
 [-7.  0.  1.]]
```

question 5

[23]:
```
# produit matriciel
D = np.dot(B,A)
print("D = ", D)
```

```
D =  [[-16.  19.  -2.  89.]
 [-65.  20.   5. 205.]
 [ -1.   5.  -1.  16.]]
```

question 6

[24]:
```
# somme matricielle
Y = np.sum(D,axis=1)
print("Y = ", Y)
```

```
Y =  [ 90. 165.  19.]
```

## 3 Exercice 2

question 1

[31]:
```
A = np.array([[4,5,6,-1],[5,10,15,2],[6,15,1,4],[-1,2,4,-2]])
vap,vecp = np.linalg.eig(A)
print("valeurs propres : ",vap)
print("vecteurs propres : ", vecp)
```

```
valeurs propres :  [ 24.49531701 -11.04368827   2.31528872  -2.76691746]
vecteurs propres :  [[ 0.340821    0.16909822  0.86888096  0.31669026]
 [ 0.73157879  0.50754043 -0.24619324 -0.382863  ]
 [ 0.57611269 -0.80330147 -0.10808058  0.10544909]
 [ 0.12933558  0.26175418 -0.41563609  0.86139636]]
```

question 2

[32]:
```
invvap = np.diag(1./vap)
invA1 = np.dot(vecp,np.dot(invvap,vecp.T))
print("invA1 = ", invA1)
invA2 = np.linalg.inv(A)
```

```
print("invA2 = ", invA2)
```

```
invA1 =  [[ 0.29197923 -0.04616272 -0.03231391 -0.25678015]
 [-0.04616272 -0.02827467  0.08020773  0.15522216]
 [-0.03231391  0.08020773 -0.04385459  0.00865551]
 [-0.25678015  0.15522216  0.00865551 -0.19907675]]
invA2 =  [[ 0.29197923 -0.04616272 -0.03231391 -0.25678015]
 [-0.04616272 -0.02827467  0.08020773  0.15522216]
 [-0.03231391  0.08020773 -0.04385459  0.00865551]
 [-0.25678015  0.15522216  0.00865551 -0.19907675]]
```

## 4 Exercice 3

question 1

[33]:
```
A = np.array([[1, -1,2 ,1 ,2],[-1,2,3,-4,1],[0,-1,1,0,0]])
rankA = np.linalg.matrix_rank(A)
print("rankA = ", rankA)
```

```
rankA =  3
```

question 2

[34]:
```
b = np.array([[3],[-7],[1]])
Xs = np.linalg.solve(A[0:3,0:3],b)
print("sol de Ax=b : ", Xs)
```

```
sol de Ax=b :  [[ 2.5]
 [-1.5]
 [-0.5]]
```

## 5 Exercice 4

[37]:
```
A = np.array([(4, 4, 8), (1, -3, 5), (3, 7, 3)], float)
B = np.array([0, 1, -1], float).reshape(-1, 1)
C = np.concatenate((A, B), axis=1)
print("A = ",A)
print("B = ",B)
print("C = ",C)
```

```
A =  [[ 4.  4.  8.]
 [ 1. -3.  5.]
 [ 3.  7.  3.]]
B =  [[ 0.]
 [ 1.]
 [-1.]]
C =  [[ 4.  4.  8.  0.]
 [ 1. -3.  5.  1.]
 [ 3.  7.  3. -1.]]
```

## 6 Exercice 5

question 1

[2]:
```
A=np.array([[4,2,1,4,5,6,7],
   [1,4,-6,2,2,0,-2],
   [1,-2,3,3,-5,6,1],
   [1,-1,1,-1,2,-3,-1],
   [1,1,0,8,2,3,4]])
b=np.array([[1],[2],[-10],[-2],[3]])
print("A = ",A)
print("b = ",b)
```

```
A =  [[ 4  2  1  4  5  6  7]
 [ 1  4 -6  2  2  0 -2]
 [ 1 -2  3  3 -5  6  1]
 [ 1 -1  1 -1  2 -3 -1]
 [ 1  1  0  8  2  3  4]]
b =  [[  1]
 [  2]
 [-10]
 [ -2]
 [  3]]
```

question 2

[3]:
```
G = np.hstack((A,b))
print("G = ",G)
```

```
G =  [[  4   2   1   4   5   6   7   1]
 [  1   4  -6   2   2   0  -2   2]
 [  1  -2   3   3  -5   6   1 -10]
 [  1  -1   1  -1   2  -3  -1  -2]
 [  1   1   0   8   2   3   4   3]]
```

question 3

[84]:
```
def permL(G,i,j):
    C = np.copy(G)
    a = np.copy(C[i,:])
    C[i,:] = np.copy(C[j,:])
    C[j,:] = a
    return C
```

question 4

[83]:
```
def ajoutS(G,i,j,a):
    C = np.copy(G)
    C[i,:] = np.copy(C[i,:])+a*np.copy(C[j,:])
    return C
```

question 5

[85]:
```
def multiS(G,i,a):
    C = np.copy(G)
    C[i,:] = a*np.copy(C[i,:])
    return C
```

question 6

[10]:
```
print(permL(G,1,2))
print(ajoutS(G,0,1,2))
print(multiS(G,0,3))
```

```
[[  4   2   1   4   5   6   7   1]
 [  1  -2   3   3  -5   6   1 -10]
 [  1   4  -6   2   2   0  -2   2]
 [  1  -1   1  -1   2  -3  -1  -2]
 [  1   1   0   8   2   3   4   3]]
[[  6  10 -11   8   9   6   3   5]
 [  1   4  -6   2   2   0  -2   2]
 [  1  -2   3   3  -5   6   1 -10]
 [  1  -1   1  -1   2  -3  -1  -2]
 [  1   1   0   8   2   3   4   3]]
[[ 12   6   3  12  15  18  21   3]
 [  1   4  -6   2   2   0  -2   2]
 [  1  -2   3   3  -5   6   1 -10]
 [  1  -1   1  -1   2  -3  -1  -2]
 [  1   1   0   8   2   3   4   3]]
```

question 7

[122]:
```
def pivotGauss(G):
    C = np.copy(G)
    n = C.shape[0]
    m = C.shape[1]
    p = 0
    for j in range(m):
        if p < n:
            k = np.argmax(abs(C[p:n,j]) )
            k = k +p
            if C[k,j] != 0:
                C = multiS(C,k,1/C[k,j])
                print("reduit: ")
                print(str(C))
                if k != p:
                    C = permL(C,k,p)
                    print("permut:")
                    print(str(C))
                for i in range(1,n):
```

```
                        if i>p:
                            C = ajoutS(C,i,p,-C[i,j])
                            print("echel:")
                            print(str(C))
                p = p+1
    return C
```

question 8

[123]:
```
print(pivotGauss(G))
```

```
reduit:
[[  1   0   0   1   1   1   1   0]
 [  1   4  -6   2   2   0  -2   2]
 [  1  -2   3   3  -5   6   1 -10]
 [  1  -1   1  -1   2  -3  -1  -2]
 [  1   1   0   8   2   3   4   3]]
echel:
[[  1   0   0   1   1   1   1   0]
 [  0   4  -6   1   1  -1  -3   2]
 [  1  -2   3   3  -5   6   1 -10]
 [  1  -1   1  -1   2  -3  -1  -2]
 [  1   1   0   8   2   3   4   3]]
echel:
[[  1   0   0   1   1   1   1   0]
 [  0   4  -6   1   1  -1  -3   2]
 [  0  -2   3   2  -6   5   0 -10]
 [  1  -1   1  -1   2  -3  -1  -2]
 [  1   1   0   8   2   3   4   3]]
echel:
[[  1   0   0   1   1   1   1   0]
 [  0   4  -6   1   1  -1  -3   2]
 [  0  -2   3   2  -6   5   0 -10]
 [  0  -1   1  -2   1  -4  -2  -2]
 [  1   1   0   8   2   3   4   3]]
echel:
[[  1   0   0   1   1   1   1   0]
 [  0   4  -6   1   1  -1  -3   2]
 [  0  -2   3   2  -6   5   0 -10]
 [  0  -1   1  -2   1  -4  -2  -2]
 [  0   1   0   7   1   2   3   3]]
reduit:
[[  1   0   0   1   1   1   1   0]
 [  0   1  -1   0   0   0   0   0]
 [  0  -2   3   2  -6   5   0 -10]
 [  0  -1   1  -2   1  -4  -2  -2]
 [  0   1   0   7   1   2   3   3]]
echel:
```

```
[[  1   0   0   1   1   1   1   0]
 [  0   1  -1   0   0   0   0   0]
 [  0   0   1   2  -6   5   0 -10]
 [  0  -1   1  -2   1  -4  -2  -2]
 [  0   1   0   7   1   2   3   3]]
echel:
[[  1   0   0   1   1   1   1   0]
 [  0   1  -1   0   0   0   0   0]
 [  0   0   1   2  -6   5   0 -10]
 [  0   0   0  -2   1  -4  -2  -2]
 [  0   1   0   7   1   2   3   3]]
echel:
[[  1   0   0   1   1   1   1   0]
 [  0   1  -1   0   0   0   0   0]
 [  0   0   1   2  -6   5   0 -10]
 [  0   0   0  -2   1  -4  -2  -2]
 [  0   0   1   7   1   2   3   3]]
reduit:
[[  1   0   0   1   1   1   1   0]
 [  0   1  -1   0   0   0   0   0]
 [  0   0   1   2  -6   5   0 -10]
 [  0   0   0  -2   1  -4  -2  -2]
 [  0   0   1   7   1   2   3   3]]
echel:
[[  1   0   0   1   1   1   1   0]
 [  0   1  -1   0   0   0   0   0]
 [  0   0   1   2  -6   5   0 -10]
 [  0   0   0  -2   1  -4  -2  -2]
 [  0   0   1   7   1   2   3   3]]
echel:
[[  1   0   0   1   1   1   1   0]
 [  0   1  -1   0   0   0   0   0]
 [  0   0   1   2  -6   5   0 -10]
 [  0   0   0  -2   1  -4  -2  -2]
 [  0   0   0   5   7  -3   3  13]]
reduit:
[[  1   0   0   1   1   1   1   0]
 [  0   1  -1   0   0   0   0   0]
 [  0   0   1   2  -6   5   0 -10]
 [  0   0   0  -2   1  -4  -2  -2]
 [  0   0   0   1   1   0   0   2]]
permut:
[[  1   0   0   1   1   1   1   0]
 [  0   1  -1   0   0   0   0   0]
 [  0   0   1   2  -6   5   0 -10]
 [  0   0   0   1   1   0   0   2]
 [  0   0   0  -2   1  -4  -2  -2]]
echel:
```

```
[[  1   0   0   1   1   1   1   0]
 [  0   1  -1   0   0   0   0   0]
 [  0   0   1   2  -6   5   0 -10]
 [  0   0   0   1   1   0   0   2]
 [  0   0   0   0   3  -4  -2   2]]
reduit:
[[  1   0   0   1   1   1   1   0]
 [  0   1  -1   0   0   0   0   0]
 [  0   0   1   2  -6   5   0 -10]
 [  0   0   0   1   1   0   0   2]
 [  0   0   0   0   1  -1   0   0]]
[[  1   0   0   1   1   1   1   0]
 [  0   1  -1   0   0   0   0   0]
 [  0   0   1   2  -6   5   0 -10]
 [  0   0   0   1   1   0   0   2]
 [  0   0   0   0   1  -1   0   0]]
```