```python
#%% TP 11 corrigé

import numpy as np

#%% exo 1

# q1
A1 = np.array([[2,4,1],[5,3,6]])

# q2
A1[0,0]=20

# q3
# B1 est la matrice identité de taille 5
# B2 est la matrice diagonale de coefficients diagonaux 4,2,7,1,2
# B3 est la matrice diagonale de coefficients diagonaux -2,0,-5,1,0
# B4 est la matrice de taille 2 x 3 ne contenant que des 5
# B5 renvoie un message d'erreur

# q4
C1 = 7*np.ones((2,2))
C2 = 3*np.ones((3,3)) + np.eye(3)
C3 = np.ones((4,4)) + np.diag([4,7,-1,1])

#%% exo 2

# q1
def carree(A):
    return np.size(A,0) == np.size(A,1)
# attention : évitez d'écrire "if bool : return True, else : return False"

# q2
def size_prod(A,B):
    nA = np.size(A,0)
    pA = np.size(A,1)
    nB = np.size(B,0)
    pB = np.size(B,1)
    if pA == nB:
        return nA,pB
    else:
        return "produit impossible"

# q3
def top_plus(A,x):
    A[0,0] = A[0,0]+x
    return A

# q4
def bottom_plus(A,x):
    n = np.size(A,0)
    p = np.size(A,1)
    A[n-1,p-1] = A[n-1,p-1]+x
    return A

#%% exo 3

# q1
def compte(A):
    c = 0
    n = np.size(A,0)
    p = np.size(A,1)
    for i in range(n):
        for j in range(p):
            if A[i,j] == 0:
                c = c+1
    return c

# q2
def trace(A):
    S = 0
    n = np.size(A,0) # = np.size(A,1) par hypothèse
    for k in range(n):
        S = S+A[k,k]
    return S

#%% exo 4
```

```python
# q1
def topleft(n):
    A = np.zeros((n,n))
    A[0,0]=1
    return A

# q2
def topright(n):
    A = np.zeros((n,n))
    A[0,n-1] = 1
    return A

# q3
def repete(L,p):
    n = len(L)
    A = np.zeros((n,p))
    for i in range(n):
        for j in range(p):
            A[i,j] = L[i]
    return A

#%% exo 5

# q1
def inflation(x,y):
    return 100*(y-x)/x

# q2
def list_inflation(prix):
    n = np.size(prix,1) # nombre d'articles
    L = []
    for k in range(n):
        t = inflation(prix[0,k],prix[1,k])
        L.append(t)
    return L

# q3
def inflation_1(prix):
    L = list_inflation(prix)
    return sum(L)/len(L)

# q4
def inflation_2(prix):
    total_2023 = sum(prix[0,:])
    total_2024 = sum(prix[1,:])
    # on peut aussi faire des boucles for pour calculer ces sommes
    return inflation(total_2023,total_2024)

# q5 : les deux méthodes ne mènent pas au même résultat.
# Pour fixer les idées, prenons un cas extrême où il n'y a
# que deux articles : un à 1 euros ne subissant pas d'augmentation
# et un à 100 euros subissant 100% d'augmentation.
# La méthode 1 conclut à une inflation moyenne de 50%.
# La méthode 2 conclut à une inflation proche de 100%.
# La méthode 2 semble plus réaliste car il n'y a pas de sens
# à faire la moyenne de pourcentage d'augmentation sur des articles
# de prix différents. Pourtant, on peut aussi estimer que les articles
# peu chers sont souvent ceux achetés plus souvent. Dans l'exemple
# ci-dessus, si l'article à 1 euros est acheté 100 fois plus souvent
# que l'article à 100 euros, alors il y a bien un sens à parler d'une
# inflation moyenne de 50%.
# Bref, calculer l'inflation, ce n'est pas si simple...

#%% exo 6

# q1
# appliquée avec des matrices, le test == ne renvoie pas un seul
# booléen, mais un tableau de booléens.

# q2
# Lorsque les tailles sont différentes, on obtient cette fois-ci
# un seul booléen, False (ainsi qu'un message d'erreur).

# q3
def egal(A,B):
```

```
    nA = np.size(A,0)
    pA = np.size(A,1)
    nB = np.size(B,0)
    pB = np.size(B,1)
    if nA == nB and pA == pB:
        for i in range(nA):
            for j in range(pA):
                if A[i,j]!=B[i,j]:
                    return False
        return True
    else :
        return False

# on peut aussi proposer :

def egal_bis(A,B):
    nA = np.size(A,0)
    pA = np.size(A,1)
    nB = np.size(B,0)
    pB = np.size(B,1)
    if nA == nB and pA == pB:
        res = True
        for i in range(nA):
            for j in range(pA):
                res = res and A[i,j]==B[i,j]
        return res
    else :
        return False

#%% exo 7

# q1
def positifs(A):
    n = np.size(A,0)
    p = np.size(A,1)
    for i in range(n):
        for j in range(p):
            if A[i,j]<0:
                return False
    return True

# q2
def stochastique(A):
    if positifs(A):
        n = np.size(A,0)
        for i in range(n):
            if sum(A[i,:])!=1:
                return False
        return True
    else :
        return False

# q3
def bistochastique(A):
    B = np.transpose(A)
    return stochastique(A) and stochastique(B)

# q4
def stochastique_bis(A):
    n = np.size(A,0)
    p = np.size(A,1)
    Up = np.ones((p,1))
    Un = np.ones((n,1))
    AU = np.dot(A,Up)
    return egal(AU,Un) and positifs(A)

# q5
def bistochastique_bis(A):
    n = np.size(A,0)
    p = np.size(A,1)
    if n==p :
        U = np.ones((n,1))
        AU = np.dot(A,U)
        B = np.transpose(A)
        BU = np.dot(B,U)
        return egal(AU,U) and egal(BU,U) and positifs(A)
```

```
    else :
        return False # nb : une matrice bisto est forcément carrée

#%% exo 8

# q1
def determinant(A):
    return A[0,0]*A[1,1]-A[0,1]*A[1,0]

# q2
def inverse(A):
    det = determinant(A)
    if det!=0:
        a = A[0,0]
        b = A[0,1]
        c = A[1,0]
        d = A[1,1]
        return (1/det)*np.array([[d,-b],[-c,a]])
    else :
        return "la matrice n'est pas inversible"

# q3
# on choisit A, on demande B = inverse(A) et on
# vérifie que np.dot(A,B) renvoie la matrice identité

# q4
def echange(A,i1,i2):
    p = np.size(A,1)
    for j in range(p):
        save = A[i1,j] # on sauvegarde A[i1,j]
        A[i1,j] = A[i2,j] # on écrase A[i1,j]
        A[i2,j] = save # on utilise la sauvegarde pour modifier la ligne i1
    return A

# q5
# cette question est hors programme en BCPST.
```