```
# ========================== #
# ========================== #
# Module de calcul matriciel #
# JANVIER 2016               #
# Contenu :                  #
#   (*) Transposition        #
#   (*) Affinité             #
#   (*) Transvection         #
#   (*) Echelonnage          #
# ========================== #
# ========================== #

# --------------------- #
# --> TRANSPOSITION <-- #
# --------------------- #

# N'importons de numpy que ce dont nous avons réellement besoin !
from numpy import array, identity, zeros, dot

def LS_build(n,l1,l2):
    """
    Fonction de création d'une matrice de TRANSPOSITION.
    Dans cette fonction, on part du principe que les arguments sont cohérents.
    """
    LS = identity(n)
    LS[l1,l1], LS[l2,l2] = 0, 0
    LS[l1,l2], LS[l2,l1] = 1, 1
    # Renvoi du tableau LS.
    return LS

def LineSwap1(M,l1,l2):
    """
    Fonctions permettant l'échange de la ligne l1 et de la ligne l2 d'une
    matrice M. Cet échange se fait grâce à un produit matriciel idoine et crée
    une nouvelle matrice.
    On effectue l'échange lorsque :
        (1) la matrice M comporte au moins deux lignes.
        (2) les indices l1 et l2 sont positifs et strictement inférieurs au
            nombre de lignes de M.
        (3) l1 est différent de l2.
    On pourra compléter le code lorsqu'au moins l'une de ces conditions n'est
    pas satisfaite.
    """
    n = M.shape[0]
    if n >= 2 and l1 >= 0 and l1 < n and l2 >= 0 and l2 < n and l1 != l2:
        # Renvoi du produit matriciel.
        return dot(LS_build(n,l1,l2),M)

def LineSwap2(M,l1,l2):
    """
    Fonctions permettant l'échange de la ligne l1 et de la ligne l2 d'une
    matrice M. Cet échange se fait SUR PLACE sans produit matriciel.
    On effectue l'échange lorsque :
        (1) la matrice M comporte au moins deux lignes.
        (2) les indices l1 et l2 sont positifs et strictement inférieurs au
            nombre de lignes de M.
        (3) l1 est différent de l2.
    On pourra compléter le code lorsqu'au moins l'une de ces conditions n'est
    pas satisfaite.
    """
    n = M.shape[0]
    if n >= 2 and l1 >= 0 and l1 < n and l2 >= 0 and l2 < n and l1 != l2:
        L = M[l1].copy()
        # On pouvait aussi utiliser : L = array(M[l1]) ou L = list(M[l1]).
        M[l1], M[l2] = M[l2], L

# ---------------- #
```

```
# --> AFFINITE <-- #
# ---------------- #

def A_build(n,l,Lambda):
    """
    Fonction de création d'une matrice d'AFFINITE.
    Dans cette fonction, on part du principe que les arguments sont cohérents.
    """
    AFF = identity(n)
    AFF[l,l] = Lambda
    # Renvoi du tableau AFF
    return AFF

def LineMult1(M,l,Lambda):
    """
    Fonctions permettant la multiplication de la ligne l par le scalaire Lambda
    d'une matrice M donnée. Cette multiplication se fait grâce à un produit
    matriciel idoine et crée une nouvelle matrice.
    On effectue la multiplication lorsque :
        (1) L'indice de ligne l est positif et strictement inférieur au nombre de
            lignes de M.
        (2) Lambda est non nul
    On pourra compléter le code lorsqu'au moins l'une de ces conditions n'est
    pas satisfaite.
    """
    n = M.shape[0]
    if l >= 0 and l < n and Lambda != 0:
        return dot(A_build(n,l,Lambda),M)

def LineMult2(M,l,Lambda):
    """
    Fonctions permettant la multiplication de la ligne l par le scalaire Lambda
    d'une matrice M donnée. Cette multiplication se fait SUR PLACE sans produit
    matriciel.
    On effectue la multiplication lorsque :
        (1) L'indice de ligne l est positif et strictement inférieur au nombre de
            lignes de M.
        (2) Lambda est non nul
    On pourra compléter le code lorsqu'au moins l'une de ces conditions n'est
    pas satisfaite.
    """
    n = M.shape[0]
    if l >= 0 and l < n and Lambda != 0:
        M[l] *= Lambda

# -------------------- #
# --> TRANSVECTION <-- #
# -------------------- #

def TV_build(n,l1,l2,Lambda):
    """
    Fonction de création d'une matrice de TRANSVECTION.
    Dans cette fonction, on part du principe que les arguments sont cohérents.
    """
    TV = identity(n)
    TV[l1,l2] = Lambda
    # Renvoi du tableau TV
    return TV

def Transvec1(M,l1,l2,Lambda):
    """
    Fonctions permettant la multiplication de la ligne l2 d'une matrice donnée M
    par le scalaire Lambda et l'addition du résultat à la ligne l1 de M. Cette
    multiplication se fait grâce à un produit matriciel idoine et crée une
    nouvelle matrice.
    On effectue l'échange lorsque :
        (1) la matrice M comporte au moins deux lignes.
```

```
        (2) les indices l1 et l2 sont positifs et strictement inférieurs au
            nombre de lignes de M.
        (3) l1 est différent de l2.
    On pourra compléter le code lorsqu'au moins l'une de ces conditions n'est
    pas satisfaite.
    """
    n = M.shape[0]
    if n >= 2 and l1 >= 0 and l1 < n and l2 >= 0 and l2 < n and l1 != l2:
        return dot(TV_build(n,l1,l2,Lambda),M)

# Fonctions permettant la multiplication de la ligne l2 d'une matrice donnée M
# par le scalaire Lambda et l'addition du résultat à la ligne l1 de M.
def Transvec2(M,l1,l2,Lambda):
    """
    Fonctions permettant la multiplication de la ligne l2 d'une matrice donnée M
    par le scalaire Lambda et l'addition du résultat à la ligne l1 de M. Cette
    opération se fait SUR PLACE sans produit matriciel.
    On effectue l'échange lorsque :
        (1) la matrice M comporte au moins deux lignes.
        (2) les indices l1 et l2 sont positifs et strictement inférieurs au
            nombre de lignes de M.
        (3) l1 est différent de l2..
    On pourra compléter le code lorsqu'au moins l'une de ces conditions n'est
    pas satisfaite.
    """
    n = M.shape[0]
    if n >= 2 and l1 >= 0 and l1 < n and l2 >= 0 and l2 < n and l1 != l2:
        M[l1] += Lambda*M[l2]

# ------------------- #
# --> ECHELONNAGE <-- #
# ------------------- #

def Echelon_base(M):
    """
    Cette fonction échelonne la matrice M en lignes SUR PLACE.
    """
    (n,p) = M.shape
    # Cas général : M comporte au moins deux lignes (sinon, on ne fait rien !)
    if n > 1:
        LC, CC = 0, 0
        while LC < n and CC < p:
            # détermination du pivot.
            # Stratégie de choix : on retient le pivot de plus grande valeur
            # absolue.
            pivot = 0
            k = LC
            for i in range(LC,n):
                if abs(M[i,CC]) > abs(pivot):
                    pivot = M[i,CC]
                    k = i
            # Si le pivot est non nul...
            if pivot != 0:
                # si la ligne du pivot n'est pas la ligne courante on échange la
                # la ligne du pivot et la ligne courante
                if k != LC:
                    LineSwap2(M,LC,k)
                # Mise à 0 de la sous-colonne
                for i in range(LC + 1,n):
                    Transvec2(M,i,LC,-M[i,CC]/pivot)
                # On incrément l'indice de la ligne courante
                LC += 1
            # On passe à la colonne suivante
            CC += 1
    return M

def EchelonReduit_base(M):
```

```
    """
    Cette fonction échelonne et réduit la matrice M en lignes SUR PLACE.
    """
    (n,p) = M.shape
    # Cas particlier : M comporte une seule ligne
    if n == 1:
        # Détermination du premier coefficient non nul
        pivot, k = M[0,0], 0
        while M[0,k] == 0 and k<p:
            pivot = M[0,k]
            k += 1
        pivot = M[0,k]
        if pivot != 0:
            print(pivot)
            LineMult2(M,0,1/pivot)
        return M
    # Cas général : M comporte au moins deux lignes
    else:
        LC, CC = 0, 0
        while LC < n and CC < p:
            # détermination du pivot
            pivot = 0
            k = LC
            for i in range(LC,n):
                if abs(M[i,CC]) > abs(pivot):
                    pivot = M[i,CC]
                    k = i
            # Si le pivot est non nul...
            if pivot != 0:
                # si la ligne du pivot n'est pas la ligne courante on échange la
                # la ligne du pivot et la ligne courante.
                if k != LC:
                    LineSwap2(M,LC,k)
                # Réduction (on normalise la ligne du pivot en la divisant par
                # le pivot !
                LineMult2(M,LC,1/pivot)
                # Mise à 0 des coefficients de la colonne CC grâce à des
                # transvections sur les lignes d'indices i différents de LC.
                for i in range(n):
                    if i != LC:
                        Transvec2(M,i,LC,-M[i,CC])
                # On incrément l'indice de la ligne courante.
                LC += 1
            # On passe à la colonne suivante.
            CC += 1
        return M

def Echelon(M):
    """
    Cette fonction échelonne la matrice M en lignes SUR PLACE.
    Elle renvoie un tuple de trois éléments :
        (*) La matrice M échelonnée
        (*) Le produit des matrices de transvection/transposition
        (*) Un booléen valant True/False suivant que M est ou non inversible
    """
    (n,p) = M.shape
    # Cas particulier : M ne comporte qu'une seule ligne.
    if n == 1:
        P = identity(1)
        # Il y a au moins deux colonnes.
        if p > 1:
            return(M,P,False)
        # Il y a une seule colonne.
        else:
            if M[0,0] == 0:
                return(M,P,False)
            else:
```

```
                return(M,P,True)
    # Cas général : M comporte au moins deux lignes.
    else:
        P = identity(n)
        Inversible = True
        LC, CC = 0, 0
        while LC < n and CC < p:
            # détermination du pivot.
            pivot = 0
            k = LC
            for i in range(LC,n):
                if abs(M[i,CC]) > abs(pivot):
                    pivot = M[i,CC]
                    k = i
            # Si le pivot est non nul.
            if pivot != 0:
                # si la ligne du pivot n'est pas la ligne courante on échange la
                # la ligne du pivot et la ligne courante.
                if k != LC:
                    P = dot(LS_build(n,LC,k),P)
                    LineSwap2(M,LC,k)
                # Mise à 0 de la sous-colonne.
                for i in range(LC + 1,n):
                    Lambda = -M[i,CC]/pivot
                    P = dot(TV_build(n,i,LC,-Lambda),P)
                    Transvec2(M,i,LC,Lambda)
                # On incrément l'indice de la ligne courante.
                LC += 1
            else:
                # Dès que l'un des pivot est nul, la matrice M n'est pas
                # inversible.
                Inversible = False
            # On passe à la colonne suivante.
            CC += 1
        return(M,P,Inversible)

"""
Quelques matrices pour tester, en particulier, les programmes d'échelonnage.
Les calculs se font systématiquement avec des coefficients matriciels de type
"flottant".
"""

A = array([[1,2,3],[4,6,8],[11,20,56]],dtype=float)

B = array([[1,2,3],[0,6,8],[0,0,56]],dtype=float)

C = array([[1,2,2,-3,2],[2,4,1,0,-5],[4,8,5,-6,-1],[-1,-2,-1,1,1]],dtype=float)

D = array([[2,-1,0],[-1,2,-1],[0,-1,2]],dtype=float)

E = array([[2,-3,1,-1],[-2,2,-3,2],[4,-9,-2,3],[-2,5,5,-4]],dtype=float)

F = array([[1,2],[4,6]],dtype=float)

G = array([[3,2,-1,15],[2,3,1,10],[1,1,1,6]],dtype=float)

H = array([[0,0,0,0,4],[0,0,0,1,0],[0,0,1,0,0],[0,5,0,0,0],[1,0,0,0,0]],dtype=float)

I = array([[0,0,0,0,2,-4,11,5,2,0,2,55]],dtype=float)

J = array([[5]],dtype=float)

K = array([[0]],dtype=float)
```