Calcul matriciel - Corrigé

```python
from copy import *
eps = 1e-12 # valeur à partir de laquelle un terme sera considéré nul

# on crée ici une nouvelle exception pour gérer les erreurs sur les matrices
class ExceptionMatrices(Exception):
    def __init__(self, raison):
        self.raison = raison
    def __str__(self):
        return self.raison

def cree_matrice(p, q, valeur=0):
    # crée une matrice de type (p,q)
    # où chaque élément vaut valeur (0 par défaut)
    M = [None] * p
    for i in range(p):
        # ligne i
        M[i] = [valeur] * q
    return M

def identite(n):
    # matrice identité d'ordre n
    M = cree_matrice(n, n, 0)
    for i in range(n):
        M[i][i] = 1
    return M

def nblignes(M):
    # renvoie le nombre de lignes de la matrice
    return(len(M))

def nbcolonnes(M):
    # renvoie le nombre de colonnes de la matrice
    return(len(M[0]))

def dimensions(M):
    # taille de la matrice
    return (nblignes(M), nbcolonnes(M))

def ecrit_matrice(A):
    # affichage de la matrice A
    p, q = dimensions(A)
    for i in range(p):
        s = ''
        for j in range(q):
            if A[i][j] >=0:
                s = s + ' '
            s = s + '{:.3f}'.format(A[i][j])+'\t'
        print(s)
    print()

def transpose(M):
    # retourne la transposée de M en utilisant
    # la définition d'une liste par compréhension
    p, q = dimensions(M)
    return [[M[i][j] for i in range(p)] for j in range(q)]
    # attention à l'ordre des indices!

```

```python
def transpose2(M):
    # retourne la transposé, version plus élémentaire
    p, q = dimensions(M)
    T = cree_matrice(q, p, None)
    for i in range(p):
        for j in range(q):
            T[j][i] = M[i][j]
    return T

def prod_scal(k, M):
    # produit de la matrice M par le scalaire k
    p, q = dimensions(M)
    return [[k * M[i][j] for j in range(q)] for i in range(p)]

def somme(A, B):
    # somme de deux matrices
    p, q = dimensions(A)
    p1, q1 = dimensions(B)
    if p != p1 or q != q1:
        raise ExceptionMatrices('Tailles incompatibles dans la somme.')
    return [[A[i][j] + B[i][j] for j in range(q)] for i in range(p)]

def prod(A, B):
    # produit de deux matrices, version basique
    p, q = dimensions(A)
    q1, r = dimensions(A)
    if q != q1:
        raise ExceptionMatrices('Tailles incompatibles dans le produit.')
    C = cree_matrice(p, r, 0)
    for i in range(p):
        for k in range(r):
            for j in range(q):
                C[i][k] += A[i][j] * B[j][k]
    return C

def entre_systeme():
    # entrée du système par l'utilisateur, présentation rudimentaire mais suffisante
    # cette procédure renvoie la matrice complète du système, avec second membre
    p = int(input("Nombre d'équations ?"))
    q = int(input("Nombre d'inconnues ?"))
    A = cree_matrice(p, q+1, 0)
    for i in range(p):
        if i == 0:
            print('Entrée de la 1ère ligne de la matrice du système:')
        else:
            print('Entrée de la ',i+1, 'ème ligne de la matrice du système:')
        for j in range(q):
            print('A[{:d},{:d}]= '.format(i+1,j+1), end='')
            A[i][j] = float(input())
    print('Entrée du second membre:')
    for i in range(p):
        print('B[{:d}]= '.format(i+1), end='')
        A[i][q] = float(input())
    return A

def ecrit_systeme(A):
    # affichage du système, présentation rudimentaire
    p, q = dimensions(A)
    for i in range(p):
        for j in range(q-1):
            print('\t{:+.3f}*x[{:d}]'.format(A[i][j],j+1),end='')
```

```
            if j == q-2:
                print('\t=',A[i][q-1])
    print()

def ligne_pivot(A, j):
    # cas d'une matrice carrée
    # renvoie le numéro de ligne où figure le pivot dans la jème colonne
    # en dessous de la ligne numéro j
    # renvoie -1 si pas trouvé: erreur, matrice non inversible
    n = nblignes(A)
    ligne = j # ligne du pivot provisoire
    pivot = abs(A[j][j]) # maximum provisoire
    if pivot == 1:
        return ligne
    # s'il vaut +-1, on le laisse
    for k in range(j+1, n):
        coeff = abs(A[k][j])
        if coeff > pivot:
            pivot = coeff
            ligne = k
    if pivot < eps:
        return -1
    else:
        return ligne

def ligne_pivot2(A, i, j):
    # cas d'une matrice quelconque
    # renvoie le numéro de ligne où figure le pivot quand on est
    #  ligne i colonne j (en-dessous de la ligne i)
    # renvoie -1 si pas trouvé
    n = nblignes(A)
    ligne = i # ligne du pivot provisoire
    pivot = abs(A[i][j]) # maximum provisoire
    if pivot == 1:
        return ligne
    for k in range(i+1, n):
        coeff = abs(A[k][j])
        if coeff > pivot:
            pivot = coeff
            ligne = k
    if pivot < eps:
        return -1
    else:
        return ligne

def echange_lignes(A, i, j):
    # échange dans A les lignes de numéros i et j
    # cette procédure ne renvoie rien, elle modifie A en place
    A[i], A[j] = A[j], A[i]

def transvection(A, k, i, mu):
    # fait dans A l'opération Lk <-- Lk + mu*Li
    # cette procédure ne renvoie rien, elle modifie A en place
    n = nbcolonnes(A)
    for j in range(n):
        A[k][j] += mu * A[i][j]
        if abs(A[k][j]) < eps:
            A[k][j] = 0

def mult_ligne(A, i, mu):
    # multiplie la ligne Li par mu
```

```
180     # cette procédure ne renvoie rien, elle modifie A en place
181     n = nbcolonnes(A)
182     for j in range(n):
183         A[i][j] *= mu
184 
185 def triangularise(A):
186     # applique la méthode du pivot en utilisant les procédures précédentes
187     # pour rendre le système (carré) triangulaire
188     # cette procédure ne renvoie rien, elle modifie A en place
189     n = nblignes(A)
190     for i in range(n):
191         ligne = ligne_pivot(A, i)
192         if ligne == -1:
193             raise ExceptionMatrices('Matrice non inversible!')
194         if ligne != i:
195             echange_lignes(A, i, ligne)
196         for k in range(i+1, n):
197             coeff =  -A[k][i] / A[i][i]
198             transvection(A, k, i, coeff)
199 
200 def echelonne(A):
201     # applique la méthode du pivot en utilisant les procédures précédentes
202     # pour rendre le système echelonné (pivot total)
203     # cette procédure ne renvoie rien, elle modifie A en place
204     p, q = dimensions(A)
205     # attention: le nombre d'inconnues est q-1
206     # puisque dans la dernière colonne il y a le second membre
207     i = 0 # numéro de ligne en cours
208     j = 0 # numéro de colonne en cours
209     while i <= p-1 and j <= q-2:
210         ligne = ligne_pivot2(A, i, j)
211         if ligne == -1: # pas de pivot, on passse direct à la colonne suivante
212             j += 1
213         else:
214             if ligne != i:
215                 echange_lignes(A, i, ligne)
216             mult_ligne(A, i, 1/A[i][j])
217             A[i][j] = 1 # pour éviter les erreurs d'arrondi
218             for k in range(p):
219                 if k != i:
220                     transvection(A, k, i, -A[k][j])
221             i += 1
222             j += 1
223 
224 def resolution_Cramer(A):
225     # le système de matrice complète A étant supposé triangulaire,
226     # le résout et renvoie le vecteur solution
227     n = nblignes(A)
228     X = [None] * n
229     for i in range(n-1, -1, -1):
230         X[i] = (A[i][n] - sum(A[i][j] * X[j] for j in range(i+1, n)) )/ A[i][i]
231     return X
232 
233 def resolution_systeme(A):
234     # le système de matrice complète A étant supposé échelonné,
235     # le résout et renvoie toutes les solutions
236     # Déjà on calcule le rang de la matrice du système
237     # (sans le second membre) i.e le nombre de lignes non nulles
238     p, q = dimensions(A)
239     rang = 0
240     i = 0
```

```
    j = 0
    while i <= p-1 and j <= q-2:
        if A[i][j] != 0:
            rang += 1
            i += 1
            j += 1
        else:
            j += 1
    print('Le rang de ce système est égal à ',rang)
    # s'il y a des lignes nulles et que le second membre n'est pas nul
    # le système est incompatible
    for i in range(rang, p):
        if A[i][q-1] != 0:
            return("Le sysème n'a pas de solution.")
    if rang == q-1:
        print('Le système a une solution unique, donnée par:')
        ecrit_systeme(A)
        for i in range (q-1):
            print('x[{:d}]={:.3f}'.format(i+1,A[i][q-1]))
    else:
        print('Le système est indéterminé')
        print('Il admet une infinité de solutions données par:')
        j = 0 # indice du premier terme non nul dans la ligne
        for i in range (rang):
            while j <= q-2 and abs(A[i][j])<eps:
                j += 1
            s = 'x[{:d}]'.format(i+1) + '={:.3f}'.format(A[i][q-1])
            for k in range(j+1,q-1):
                if abs(A[i][k]) > eps:
                    s = s + '{:+.3f}*x[{:d}]'.format(-A[i][k],k+1)
            print(s)

def determinant(A):
    # applique la méthode du pivot en utilisant les procédures précédentes
    # pour rendre la matrice triangulaire puis calcule son déterminant
    n, m = dimensions(A)
    if n != m:
        raise ExceptionMatrices("Vous cherchez le déterminant d'une matrice non carrée!")
    det = 1
    for i in range(n):
        ligne = ligne_pivot(A, i)
        if ligne == -1:
            return 0
        if ligne != i:
            echange_lignes(A, i, ligne)
            det = -det
        for k in range(i+1, n):
            coeff =  -A[k][i] / A[i][i]
            transvection(A, k, i, coeff)
    for i in range(n):
        det *= A[i][i]
    return det

def mult_ligne(A, i, mu):
    # multiplie la ligne Li par mu
    # cette procédure ne renvoie rien, elle modifie A en place
    n = nbcolonnes(A)
    for j in range(n):
        A[i][j] *= mu

def inverse(A0):
```

```
    A = deepcopy(A0)
    n, m = dimensions(A)
    if n != m:
        raise ExceptionMatrices("Vous cherchez l'inverse d'une matrice non carrée!")
    B = identite(n)
    for i in range(n):
        ligne = ligne_pivot(A, i)
        if ligne == -1:
            raise ExceptionMatrices('Matrice non inversible!')
        if ligne != i:
            echange_lignes(A, i, ligne)
            echange_lignes(B, i, ligne)
        coeff = 1/A[i][i]
        mult_ligne(A, i, coeff)
        mult_ligne(B, i, coeff)
        A[i][i] = 1 # pour éviter les erreurs d'arrondi
        for k in range(n):
            if k != i:
                coeff = -A[k][i]
                transvection(A, k, i, coeff)
                transvection(B, k, i, coeff)
    return B

# Exemples d'éxécution
#A0 = entre_systeme()
A0 = [[2,2,-3,2],[-2,-1,-3,-5],[6,4,4,16]]
A = deepcopy(A0)
# on fait une copie pour ne pas modifier le système initial
triangularise(A)
X = resolution_Cramer(A)
print('Les solutions du système:, \n')
ecrit_systeme(A0)
print('sont:','\n')
for i in range (nblignes(X)):
    print('x[{:d}]={:.3f}'.format(i+1,X[i]))
print()

# A0 = entre_systeme()
A0 = [[1,2,3,4], [5,6,7,8], [9,10,11,12], [13,14,15,16]]
A = deepcopy(A0)
print('Résolution du système:')
ecrit_systeme(A)
echelonne(A)
resolution_systeme(A)
print()

A = [[2,2,-3],[-2,-1,-3],[6,4,4]]
print('Le déterminant de la matrice: \n')
ecrit_matrice(A)
print('est égal à : ', determinant(A),'\n')

A = [[2,2,-3],[-2,-1,-3],[6,4,4]]
B = inverse(A)
print("L'inverse de la matrice A  = \n")
ecrit_matrice(A)
print('est la matrice B = \n')
ecrit_matrice(B)
print('Vérification: le produit AB est égal à:')
C= prod(A, B)
ecrit_matrice(C)

```

```
363 # ci dessous, exemple d'éxécution

Les solutions du système:,

        +2.000*x[1]        +2.000*x[2]        -3.000*x[3]        = 2
        -2.000*x[1]        -1.000*x[2]        -3.000*x[3]        = -5
        +6.000*x[1]        +4.000*x[2]        +4.000*x[3]        = 16

sont:

x[1]=-14.000
x[2]=21.000
x[3]=4.000

Résolution du système:
        +1.000*x[1]        +2.000*x[2]        +3.000*x[3]        = 4
        +5.000*x[1]        +6.000*x[2]        +7.000*x[3]        = 8
        +9.000*x[1]        +10.000*x[2]        +11.000*x[3]        = 12
        +13.000*x[1]        +14.000*x[2]        +15.000*x[3]        = 16

Le rang de ce système est égal à  2
Le système est indéterminé
Il admet une infinité de solutions données par:
x[1]=-2.000+1.000*x[3]
x[2]=3.000-2.000*x[3]

Le déterminant de la matrice:

 2.000         2.000        -3.000
-2.000        -1.000        -3.000
 6.000         4.000         4.000

est égal à :  1.9999999999999956

L'inverse de la matrice A  =

 2.000         2.000        -3.000
-2.000        -1.000        -3.000
 6.000         4.000         4.000

est la matrice B =

 4.000        -10.000        -4.500
-5.000         13.000         6.000
-1.000         2.000         1.000

Vérification: le produit AB est égal à:
 1.000        -0.000        -0.000
-0.000         1.000         0.000
 0.000         0.000         1.000
```