Correction du DS 3

Informatique pour tous, première année

Julien Reichert

Question de cours

La question de cours n'est bien entendu pas corrigée ici.

Exercice 1

Adaptation (simplification) des notes de cours :

```
def echange(M,i,j):
    """Opération élémentaire M[i] <-> M[j]"""
    tampon = M[i].copy()
    M[i] = M[j]
    M[j] = tampon

def divise(M,i,l=None):
    """Opération élémentaire M[i] <- M[i] / l"""
    for k in range(len(M[i])):
        M[i,k] /= l
```

Exercice 2

On remplace la fonction `cherche_pivot` par le code suivant :

```
def cherche_pivot(M,i):
    """Ligne >= i dont l'élément de colonne i est maximal en valeur absolue."""
    ind_max = i
    for ind in range(i,len(M)):
        if abs(M[ind,i]) > abs(M[ind_max,i]):
            ind_max = ind
    assert M[ind_max,i] != 0., "Matrice non inversible"
    return ind
```

Exercice 3

Simplification de l'algorithme de base de recherche des occurrences d'un élément, ici précisé.

```
def zerosdansliste(l):
    reponse = []
    for i in range(len(l)):
        if l[i] == 0:
            reponse.append(i)
    return reponse
```

La version sur les listes de listes nécessite simplement de faire un parcours à l'aide d'une double boucle.

```
def zerosdanslisteliste(l):
    reponse = []
    for i in range(len(l)):
        for j in range(len(l[i])):
            if l[i][j] == 0:
                reponse.append((i,j)) # parenthèses pour la fonction et pour le couple
    return reponse
```

Exercice 4

Structure choisie : tableau (`array`).

```
from numpy import array # ou matrix, plus respectueux de l'énoncé

def matrice_absolue(m, n):
    return array([[abs(j-i) for j in range(n)] for i in range(m)]) # ou matrix, donc
```

Bien entendu, il est toujours possible de créer un tableau de bonne taille (fonction `zeros` du module `numpy` ou initialisation à partir d'une liste de la bonne forme et de la fonction array) et de faire une double boucle.

On évitera cependant d'utiliser la fonction `numpy.eye` et d'additionner les matrices pseudo-diagonales obtenues, c'est peut-être facile mais bien trop lourd en complexité.

Exercice 5

```
def evalue_poly(l,x):
    xpuissancen = 1
    somme = 0
    for i in range(len(l)):
        somme += l[i] * xpuissancen # plus efficace que l[i] * x ** i
        xpuissancen *= x
    return somme

def convertit_fonction(l):
    def fonction(x):
        return evalue_poly(l,x) # ou coller le code de la fonction
    return fonction
```

```
def derive_poly(l):
    if len(l) == 1:
        return [0] # Au pire, si on assimilait [] au polynôme 0, il n'y aurait pas de contradiction
    ll = []
    for i in range(1,len(l)):
        ll.append(l[i]*i)
    return ll

def newton_poly(l,dep,eps):
    ll = derive_poly(l)
    xn = depart
    xnplusun = xn - evalue_poly(l,xn) / evalue_poly(ll,xn)
    while abs(xnplusun - xn) > epsilon:
        xn = xnplusun
        xnplusun = xnplusun - evalue_poly(l,xnplusun) / evalue_poly(ll,xnplusun)
    return xnplusun

def newton_poly_bis(l,dep,eps):
    fonction = convertit_fonction(l)
    derivee = convertit_fonction(derive_poly(l))
    xn = depart
    xnplusun = xn - fonction(xn) / derivee(xn)
    while abs(xnplusun - xn) > epsilon:
        xn = xnplusun
        xnplusun = xnplusun - fonction(xnplusun) / derivee(xnplusun)
    return xnplusun
```