Université de Strasbourg
Cours Calcul Scientifique 2
Année 2018-2019

TP 1 : Matrices creuses et factorisation LU

L'objectif de ce TP est de se familiariser avec les structures creuses et leurs utilisation dans scipy. Commencez par importer les modules python :

```
import numpy as np
import scipy as sp
import scipy.sparse as spsp
```

Partie 1. (Format Compressed Sparse Row) Nous pouvons définir une matrice grâce aux commandes suivantes:

```
row = sp.array([0, 0, 1, 2, 2, 2])
col = sp.array([0, 2, 2, 0, 1, 2])
data = sp.array([1, 2, 3, 4, 5, 6])

A = spsp.csr_matrix((data, (row, col)), shape=(3, 3))
```

1. Le format CSR correspond aux attributs `A.data`, `A.indptr`, `A.indices`. A quoi correspondent-ils ? On pourra s'aider de la commande `A.toarray()`. Comment est définie la fonction `print` ?
2. Que renvoie la commande $A[0, :]$? La notice de Scipy indique qu'elle est plus rapide que la commande $A[:, 0]$: donner une explication.
3. Peut-on peut ajouter deux matrices stockées sous format CSR ? Commentez.
4. Ecrire une fonction `matvect_multiply(A,b)`, qui prend en entrée une matrice A sous format CSR et un vecteur b et qui renvoie le vecteur $y = Ab$. **Ne parcourir que les éléments non nuls de A.** Tester votre fonction en prenant un vecteur b avec des coefficients choisis aléatoirement et en comparant le résultat avec la commande `A.dot(b)` ou encore `A@b` (que se passe-t-il avec la commande `np.dot(A,b)` ?).

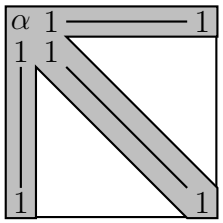
Partie 2. (Factorisation LU)

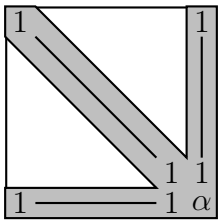
5. Ecrire une fonction `Facto_LU(A)` qui prend en entrée une matrice A et qui renvoie la factorisation LU de la matrice. La factorisation doit être "inplace", ce qui signifie que la factorisation doit être stockée à la place de A. Indication: on pourra ici travailler directement sur les lignes $A[i, :]$ (le remplissage de la matrice pourra ainsi augmenter). Ajouter un message d'erreur dès qu'un pivot est nul.
6. Ecrire une fonction `solve_LU(A,b)` qui prend en entrée une matrice A et un vecteur b et qui renvoie la solution du système linéaire $Ax = b$. Pour les algorithmes de remontée et de descente, **ne parcourir que les éléments non nuls de L et U.**
7. Testez votre méthode sur la matrice suivante :

   ```
   A = spsp.diags([- np.ones(n-1), 2*np.ones(n), -np.ones(n-1)], [-1, 0, 1])
   A = A.tocsr()
   ```

   Vérifier votre résultat avec une commande scipy.

Partie 3 (Remplissage - fill in). Nous considérons à présent les deux matrices suivantes:

$A =$ 

$,B =$ 

8. Déterminer α de telle sorte que les matrices ci-dessus aient des factorisation LU.
9. Créer les matrices précédentes à l'aide du format de matrice `dok` (dictionnary of keys), puis les convertir au format CSR.
10. A l'aide de la commande `spy` de matplotlib, observez la structure des matrices avant et après avoir effectué la factorisation LU. Commentez.

Partie 4 (temps de calcul). Nous considérons à présent une matrice bande de taille $n \times n$ avec $n = d^2$, dont la diagonale n^o 0 est constituée de 4 et les diagonales n^o 1, -1, d et $-d$ sont emplies de -1.

11. Assembler cette matrice à l'aide du format `spdiags`, puis la convertir au format CSR.
12. Grâce au module `time`, comparer les temps de calcul de votre fonction `LU` appliquée à cette matrice et à sa version dense.

```
deb = time.time()
LU(A)
fin = time.time()
print("(LU) temps de calcul = ", fin-deb)
```

13. Reprenez la question précédente avec la fonction la fonction `splu` de scipy. Commentez.