Numpy - Exercices

Voici une série de tests pour vérifier ses connaissances en Numpy. Le but est que cette bibliothèque et la programmation vectorielle qu'elle implique deviennent naturelles. Il faut oublier la programmation classique avec des boucles (il ne faut pas utiliser de boucle `for` ou autre dans cette feuille en dehors de construction de liste en compréhension).

Les fonctions ne doivent jamais modifier les arguements.

Les exercices indique le nombre de ligne de la réponse. Cela comprend l'en-tête de la fonction lorsqu'il y en a une.

Pensez à tester vos résultats.

```
import numpy as np
rand = np.random.RandomState(123)

A = np.arange(12).reshape(3,4)
B = rand.randint(-10, 10, size=(5,5))
print(A, '\n')
print(B)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]

[[  3  -8  -8  -4   7]
 [  9   0  -9 -10   7]
 [  5  -1 -10   4 -10]
 [  5   9   4  -6 -10]
 [  6  -6   7  -7  -8]]
```

Matrice carrée

Créer la matrice carrée 5x5 qui contient les 25 premiers entiers, de 1 à 25.

1 ligne

```
print(np.arange(1, 26).reshape(5, 5))
```

```
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]

 [11 12 13 14 15]
 [16 17 18 19 20]
 [21 22 23 24 25]]
```

Norme d'un vecteur

Retourner la norme euclidienne d'un vecteur sans utiliser la fonction `norm`.

2 lignes

```
def norm(v):
    return max(abs(v))
```

Sous-matrice

Extraire la sous-matrice de taille (n-2)x(m-2) qui retire les bords de la matrice A de taille nxm.

2 lignes

```
def submat(A):
      return A[1:-1, 1:-1]
print(submat(B))
```

```
     [[  0  -9 -10]
      [ -1 -10   4]
      [  9   4  -6]]
```

Vecteur aléatoire

Créer une fonction qui prend une taille n et retourne un vecteur d'entiers aléatoire de taille n compris entre -a et +a et centré le plus possible en 0.

3 lignes

```
def rand_vec(n, a):
    res =  np.random.randint(-a, a, size=n)
    return res - round(res.mean())
```

Trace

Retourner la trace d'une matrice sans utiliser la fonction `trace` (à vous de chercher les éléments qui vous manque).

2 lignes

```
def trace(A):
    return A.diagonal().sum()
```

Matrice de multiples de 3

Arrondir la matrice donnée au multiple de 3 le plus proche (pour chaque valeur).

2 lignes

```
def round3(A):
    return 3 * (A / 3).round()
```

Nombre de 9

Compter le nombre de 9 dans une matrice d'entier A donnée en paramètre.

2 lignes

```
def nb9(A):
    return (A==9).sum()
```

Colonne qui a la plus petite moyenne

Retourner l'indice de la colonne d'une matrice ayant la plus petite moyenne.

2 lignes

```
def min_col_mean(A):
      return A.mean(axis=0).argmin()
```

ChessSum

On regarde la matrice comme un échequier avec des cases blanches et des cases noires qui alternent. La case en [0,0] est noire. Calculer la somme des valeurs sur les cases blanches.

2 lignes

```
def chess_sum(A):
      return A[1::2,::2].sum() + A[::2, 1::2].sum()
```

2 minimums

Retourner les 2 plus petites valeurs d'une matrice avec une méthode de complexité linéaire. Attention, si la matrice a sa valeur minimal en double, la réponse est deux fois cette valeur minimale.

6 lignes

```
def mins(A):
    res1 = A.min()
    if (A == res1).sum() > 1:
        return res1, res1
    else:
        return res1, A[A > res1].min()
```

Lignes dans l'ordre

Ranger les lignes d'une matrice dans l'ordre croissant de leur moyenne.

2 lignes

```
def sort_lines(A):
      return A[A.sum(axis=1).argsort()]
```

Valeurs uniques

Donner la liste des valeurs uniques (qui n'apparaissent qu'une fois) dans un tableau numpy (la fonction `unique` sera utile).

3 lignes

```
def val_unique(A):
      val, nb = np.unique(A, return_counts=True)
      return val[nb == 1]
```

Tenseur magique

Constuire un tenseur 3D de taille nxnxn en utilisant n fois les n^2 premiers entiers et de sorte que la sommme des éléments de chaque plan (coupe) est toujours la même.

3 lignes

```
def mk_magic_tensor(n):
    A = np.arange(n*n).reshape([n,n])
    return np.stack([np.roll(np.roll(A, i, axis=0), i, axis=1) for i in range(n)])
```

Plans d'un tenseur

Extraire tous les plans d'un tenseur 3d. On retourne une liste de tableaux.

5 lignes (2 lignes avec la fonction `take`)

```
def extract_planes(T):
    res1 = [T[i,:,:] for i in range(T.shape[0])]
    res2 = [T[:,j,:] for j in range(T.shape[1])]
    res3 = [T[:,:,k] for k in range(T.shape[2])]
    return res1 + res2 + res3

def extract_planes(T):
    return [T.take(i, axis=d) for d in range(3) for i in range(T.shape[d])]
```

Distance entre villes

Soit une liste de points en 2D qu'on peut imaginer comme les coordonnées (x,y) de villes. On veut un tableau D qui donne la distance à vol d'oiseau entre 2 villes. Donc a D[i,j] = D[j,i] = distance entre les villes (points) i et j.

Les n points sont donnés dans une liste (n,2). Écrire la fonction qui prend ces points et retourne la matrice D des distances.

5 lignes

```
def dist_points(pts):
    P = np.array(pts)
    x = np.array([P[:,0]])  # vecteur en 2D pour pouvoir faire la transposée
    y = np.array([P[:,1]])
    return np.sqrt(np.square(x - x.T) + np.square(y - y.T))
```