TP1 Introduction à Numpy

September 30, 2019

L'objectif de ce TP est :

- d'utiliser la librairie `numpy`
- de mettre en évidence la différence de performances entre une implémentation vectorisée utilisant `numpy` et une implémentation en pur python
- de présenter le module spécifique de manipulation de graphes `networkx`

Notions visitées :

- rampe logarithmique `np.logspace`, `np.linspace`
- structure try: except
- types shape et strides
- vectorisation indexing et stacking
- broadcasting
- introduction a networkx

Pour les besoins du TP, vous importerez les modules suivants

```
In [40]: import networkx as nx # networkx (structure de graphe)
         import time   # mesure du temps
         import numpy as np # numpy
         import matplotlib.pyplot as plt # matplotlib
         import matplotlib.cm as cm  # les colormaps
         %matplotlib inline
```

1 Etude du Produit Scalaire

Soient deux vecteurs $\mathbf{x}$ et $\mathbf{y}$ deux vecteurs de $\mathbb{R}^N$.

Ecrire le produit scalaire entre ces deux vecteurs de 4 façons différentes:

1. En bouclant sur chaque élément à l'aide de la structure `for`
2. En utilisant `np.dot`
3. En utilisant `np.sum`
4. En utilisant `np.einsum`

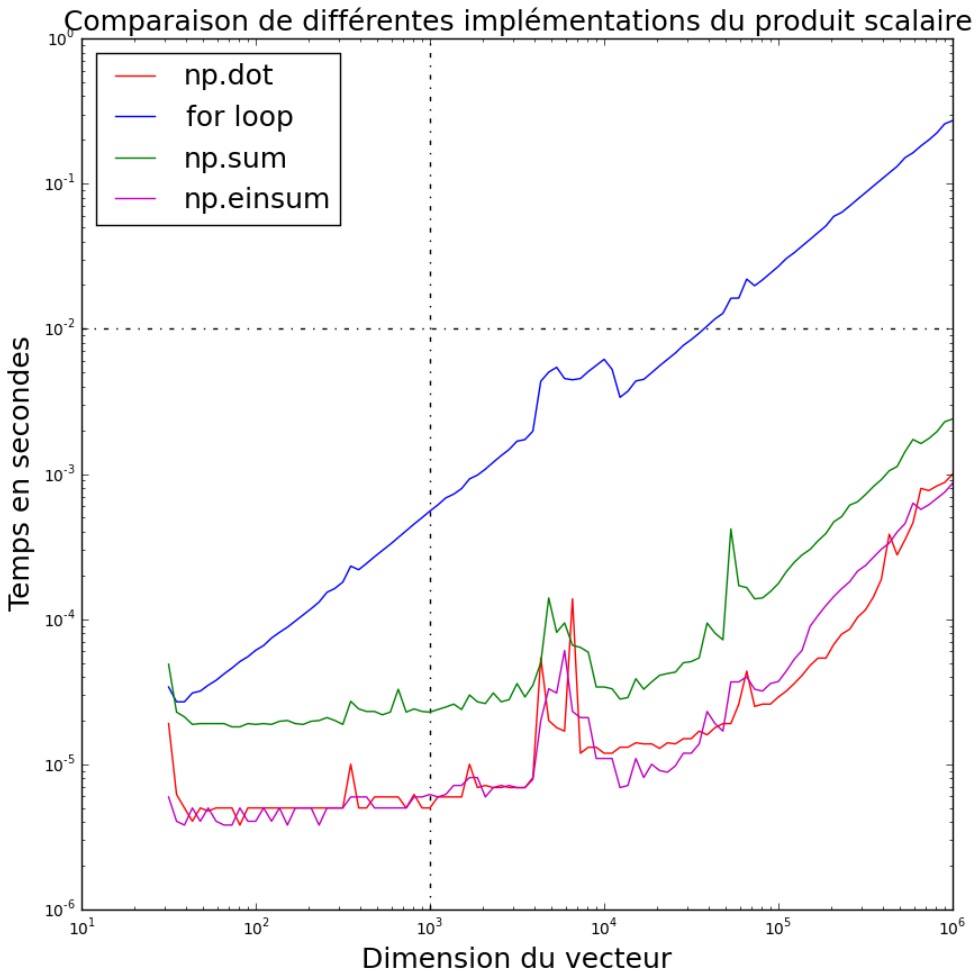
$$\mathbf{x}.\mathbf{y} = \sum_{j=0}^{N-1} x_j y_j$$

$$\mathbf{x}.\mathbf{y} = x_j y_j$$

- Comparez les performances en fonction de la valeur de N
- Conclure

La structure ci-dessous qui utilise le module `time` importé plus haut, permet de mesurer le temps d"exécution entre deux points d'une séquence d'instructions.

```
In [41]: tic = time.time()
         print tic
         time.sleep(1)
         toc = time.time()
         print toc

1410896120.15
1410896121.15

In [42]: def scal1(x,y):
             ps = np.dot(x,y)
             return(ps)

In [43]: def scal2(x,y):
             # votre code ici
             return(ps)

In [44]: def scal3(x,y):
             # votre code ici
             return(ps)

In [45]: def scal4(x,y):
             #votre code ici
             return(ps)

In [46]: tic = time.time()

In [2]: # votre code ici

In [1]: # le résultat attendu

In [52]: Image('tp1.png')

Out[52]:
```



2 Empilements de tableaux et axes

Etudier les fonctions suivantes pour générer des vecteurs et des tableaux multidimensionnels pour différents types de données numériques (entiers courts, entier longs, flottants): observer `dtype`, `shape` et `strides`:

- `np.empty()`
- `np.zeros()`
- `np.ones()`
- `np.random.rand()`
- `np.random.randn()`
- `np.eyes()`
- `np.arange` et `np.linspace`

`numpy` dispose des commandes d'empilement de vecteurs suivantes :

- `hstack`
- `vstack`
- `dstack`
- `concatenate`
- Implémenter les 3 premières fonctions à l'aide de `concatenate`.
- Vous empilerez deux vecteurs (1x3) et deux matrices (2x3).
- Précisez à chaque fois les dimensions du résultat de ces opérations.

Etudier l'opérateur:

- `np.kron`

Etudier ces propriétés à l'aide d'un bloc de données ne contenant que des valeurs `1` obtenus à l'aide de `np.ones()`.

```
In [3]: # vos expériences et commentaires ici
```

3 Type, Shape and Strides, ordre C et ordre Fortran

On considère la matrice suivante

$$\mathbf{X} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

On obtient cette matrice en créant le tableau numpy de la façon suivante :

```
In []: x = np.array([[1, 2, 3],
                    [4, 5, 6],
                    [7, 8, 9]], dtype=np.float,order='F')
```

- Visualiser le buffer de données x.data en le convertissant en chaîne de caractères.
- Etudier l'influence du paramètre `order` sur l'organisation des données en mémoire.
- Extraire la première ligne
- Extraire la deuxième colonne
- Ecrire une fonction qui extrait de `x` 4 sous-matrices (2x2)

```
In [4]: # vos expériences et commentaires ici
```

4 Broadcasting et utilisation de `axis` pour l'évaluation de fonctions multi-dimensionelles

Attention : Le terme axis est ambigue car il est utilisé par `matplotlib` et `numpy`. Dans ces deux contextes, le terme `axis` a un sens complètement différent.

- Pour `numpy` le terme `axis` désigne une dimension d'un élément tensoriel
- Pour `matplotlib` le terme `axis` désigne une classe pour gérer les axes d'un graphique
  - A ne pas confonfre avec `axes` qui désigne la zone de dessin

Utiliser le broadcasting et la syntaxe `np.newaxis` ou `None` pour évaluer la fonction

$$f(x,y) = \frac{sinx \sin y}{xy}$$

sur le pavé

$$[-3,3] \times [-3,3]$$

- Utiliser `np.linspace()`,`np.sin()`,`plt.imshow()` ou `plt.matshow()`
- Essayez différentes colormap `cm.jet`,`cm.??`,...
- Ajouter une barre de couleur
- Ecrire un programme pour évaluer quantitativement le gain de temps vis à vis d'une implémentation à l'aide de 2 boucles `for` imbriquées.

5 Un exemple d'utilisation du module `networkx`

```
In []: plt.figure(figsize=(10,10))
       import networkx as nx
```

Utilisez les fonctions

- `nx.karate_club_graph`
- `nx.spring_layout`
- `nx.draw`
- `nx.draw_networkx_nodes`
- `nx.draw_networkx_edges`
- `nx.draw_networkx_labels`

1. initialiser le graphe.
2. visualiser le graphe social.
3. Calculer la centralité des noeuds avec `nx.betweeness_centrality`
4. Trier ce dictionnaire pour déterminer les noeuds les plus centraux de ce graphe
5. Calculer la matrice d'adjacence $\mathbf{A}$ du graphe à l'aide de `nx.adjacency_matrix`
6. Visualiser cette matrice d'adjacence en noir et blanc avec `matshow`. Interpréter la signification de cette matrice.
7. Utiliser la fonction `nx.pagerank` appliquée sur ce graphe.
8. Commenter vos résultats.