Opération sur les matrices avec numpy en Python

Effectuer des calculs sur les matrices est, souvent, une opération fastidieuse et source de nombreuses erreurs. L'algorithme naïf de multiplication de matrice possède, par exemple, une complexité cubique, rendant le calcul manuel de grandes matrices vite impossible. Utiliser un ordinateur pour effectuer ce genre de calculs me semble donc une bonne chose à faire. Pour aujourd'hui, je vais vous présenter quelques fonctions en python, permettant d'effectuer des opérations sur les matrices en juste quelques lignes de codes.

Table des matières

Robin Pourtaud - Devmath - 2020-05-18

Importer numpy

Avant toute chose, pour effectuer des calculs scientifiques, vous devez importer la bibliothèque numpy.

```
import numpy as np
```

Addition et Multiplication de Matrices

```
M1 = np.array([ [1.,1.,1.],
                [1.,3.,1.],
                [1.,1.,1.]])
M2 = np.array([ [1.,1.,1.],
                [1.,2.,1.],
                [1.,1.,1.]])
MResultAdd = np.add(M1,M2)
MResultSous = np.subtract(M1,M2)
MResultMulti = M1.dot(M2)
```

Trace de Matrices

La trace d'une matrice est la somme des valeurs de la diagonale de la matrice.

```
M1 = np.array([ [1.,1.,1.],
                [1.,3.,1.],
                [1.,1.,1.]])
```

Déterminant de Matrices

Le calcul du déterminant d'une matrice est un outil nécessaire en algèbre linéaire afin de vérifier une inversibilité ou pour calculer l'inverse d'une matrice. (Wikipédia)

```
M1 = np.array([ [1.,1.,1.],
                [1.,3.,1.],
                [1.,1.,1.]])
Determinant = np.linalg.det(M1)
```

Transposée de Matrices

La transposée d'une matrice A est une matrice B telle que les colonnes et lignes de la matrice A deviennent respectivement les lignes et colonnes de la matrice B.

```
M1 = np.array([ [1.,1.,2.],
                [6.,3.,1.],
                [1.,1.,1.]])
Transpose = M1.transpose()
```

Inverse de Matrices

L'inverse de matrice prend toute son utilité dans le calcul de systèmes linéaires. Si $Ax = B$, alors $x = A^{-1}B$.

Une matrice A admet une inverse si et seulement si son déterminant est différent de 0. Il est donc important de le tester au préalable :).

```
M1 = np.array([ [1.,1.,1.],
                [1.,3.,2.],
                [1.,5.,1.]])
Inverse = np.linalg.inv(M1)
```

Valeurs propres et Vecteurs propres de Matrices

Veuillez bien prendre en compte le fait que les vecteurs propres ne sont pas uniques, l'échelle (la longueur), le signe, ou l'ordre peuvent être différents.

```
M1 = np.array([ [1.,1.,2.],
                [6.,3.,1.],
                [1.,1.,1.]])
Eig = np.linalg.eig(M1)
```

Pour en savoir plus sur le sujet, je vous renvoie vers mon article sur les valeurs et vecteurs propres.

Décomposition de Cholesky de Matrices

La décomposition de Cholesky permet de décomposer une matrice symétrique définie positive en un produit d'une matrice L par sa transposée.

```
M1 = np.array([ [1,2],
                [2,7] ])
np.linalg.cholesky(M1)
```

Le résultat de np.linalg.cholesky(M1) sera cette matrice L.

Pour en savoir plus sur l'utilité et le calcul de la matrice de Cholesky, c'est ici.

Conditionnement de Matrice

Pour définir correctement le conditionnement d'une matrice, je vous suggère mon article sur ce sujet.

```
M1 = np.array([ [1,7,2,1],
                [7,5,1,5],
                [8,6,10,9],
                [7,5,9,1] ])
Cond = np.linalg.cond(M1)
```

Par défaut, np.linalg.cond **n'utilise pas la norme infinie.** Si vous souhaitez l'utiliser, je vous propose de rajouter cette option :

```
M1 = np.array([ [1,7,2,1],
                [7,5,1,5],
                [8,6,10,9],
                [7,5,9,1] ])
Cond = np.linalg.cond(M1,np.inf)
```

Pour plus de paramètres, c'est ici.

Pour aller plus loin....

Numpy propose encore un nombre très important de fonctions relatives aux matrices....

Ici se trouvent par exemple un certain nombre d'entre-elles : https://docs.scipy.org/doc/numpy/reference/routines.linalg.html

Sources :

- https://docs.scipy.org/doc/numpy/reference/routines.linalg.html?highlight=determinant
- https://www.researchgate.net/post/Why_eigenvectors_seem_incorrect_in_python