Tableau Comparatif des Commandes pour les Matrices en Python (NumPy) et R

Opération	Python (NumPy)	R
Créer une matrice	`np.array([[1, 2], [3, 4]])`	`matrix(c(1, 2, 3, 4), nrow=2)`
Matrice nulle	`np.zeros((2, 2))`	`matrix(0, nrow=2, ncol=2)`
Matrice identité	`np.eye(3)`	`diag(3)`
Matrice remplie de 1	`np.ones((2, 2))`	`matrix(1, nrow=2, ncol=2)`
Dimensions de la matrice	`matrice.shape`	`dim(matrice)`
Transposition	`matrice.T`	`t(matrice)`
Addition de matrices	`matrice1 + matrice2`	`matrice1 + matrice2`
Multiplication élément par élément	`matrice1 * matrice2`	`matrice1 * matrice2`
Multiplication de matrices	`np.dot(matrice1, matrice2)`	`matrice1 %*% matrice2`
Inversion de matrice	`np.linalg.inv(matrice)`	`solve(matrice)`
Déterminant	`np.linalg.det(matrice)`	`det(matrice)`
Trace de la matrice	`np.trace(matrice)`	`sum(diag(matrice))`
Résolution de système linéaire	`np.linalg.solve(A, b)`	`solve(A, b)`
Valeurs propres	`np.linalg.eig(matrice)`	`eigen(matrice)`
Vecteurs propres	`np.linalg.eig(matrice)[1]`	`eigen(matrice)$vectors`
Rang de la matrice	`np.linalg.matrix_rank(matrice)`	`qr(matrice)$rank`
Norme de la matrice	`np.linalg.norm(matrice)`	`norm(matrice, type="F")`
Extraction diagonale	`np.diag(matrice)`	`diag(matrice)`
Matrice diagonale	`np.diag([1, 2, 3])`	`diag(c(1, 2, 3))`
Concaténation verticale	`np.vstack((matrice1, matrice2))`	`rbind(matrice1, matrice2)`
Concaténation horizontale	`np.hstack((matrice1, matrice2))`	`cbind(matrice1, matrice2)`
Accéder à un élément	`matrice[i, j]`	`matrice[i, j]`
Accéder à une ligne	`matrice[i, :]`	`matrice[i, ]`
Accéder à une colonne	`matrice[:, j]`	`matrice[, j]`

Résumé sur les Matrices en Python

1. Représentation des Matrices

En Python, les matrices sont généralement représentées à l'aide de la bibliothèque **NumPy**, qui fournit un objet `ndarray` pour manipuler des tableaux multidimensionnels. Une matrice est un tableau à deux dimensions.

Exemple :

```
import numpy as np
matrice = np.array([[1, 2], [3, 4]])
```

2. Opérations de Base

- **Création** : `np.array()`, `np.zeros()`, `np.ones()`, `np.eye()`.
- **Addition** : `matrice1 + matrice2`.
- **Multiplication** : `np.dot(matrice1, matrice2)` pour la multiplication matricielle, `matrice1 * matrice2` pour la multiplication élément par élément.
- **Transposition** : `matrice.T`.
- **Inversion** : `np.linalg.inv(matrice)`.
- **Déterminant** : `np.linalg.det(matrice)`.

3. Fonctions Avancées

- **Valeurs propres et vecteurs propres** : `np.linalg.eig(matrice)`.
- **Résolution de systèmes linéaires** : `np.linalg.solve(A, b)`.
- **Rang de la matrice** : `np.linalg.matrix_rank(matrice)`.
- **Norme** : `np.linalg.norm(matrice)`.

4. Manipulation des Matrices

- **Accéder aux éléments** : `matrice[i, j]`, `matrice[i, :]` (ligne), `matrice[:, j]` (colonne).
- **Concaténation** : `np.vstack()` (verticale), `np.hstack()` (horizontale).
- **Extraction diagonale** : `np.diag(matrice)`.

5. Applications

Les matrices sont utilisées dans :

- **Algèbre linéaire** : Résolution de systèmes d'équations, diagonalisation.
- **Apprentissage automatique** : Manipulation de données, calcul de distances.
- **Graphisme et Vision par Ordinateur** : Transformations géométriques, traitement d'images.

6. Avantages de NumPy

- **Efficacité** : NumPy est optimisé pour les calculs numériques.
- **Simplicité** : Syntaxe concise et intuitive.
- **Interopérabilité** : Compatible avec d'autres bibliothèques scientifiques comme SciPy, Pandas, et Matplotlib.