Informatique

11

Numpy - Array

Résumé

`import numpy as np`		

Création d'un vecteur	$F = \begin{bmatrix}1\\2\\3\end{bmatrix}$; $H = \begin{bmatrix}0\\0\\0\end{bmatrix}$
	$G = \begin{bmatrix}0\\\vdots\\10\end{bmatrix}, 100\ termes$
		`F = np.array([1,2,3])`
		`F = np.empty([1,3])` ; `V[0,:] = [1,2,3]`
		`F = np.empty([3,1])` ; `V[:,0] = [1,2,3]`
		`G = np.linspace(0,10,101)`
		`G = np.arange(0,10.1,0.1)`
		`H = np.zeros([3])` ; `H = np.zeros([3,1])`

Création d'une matrice	$K = \begin{bmatrix}0&0&0\\0&0&0\\0&0&0\end{bmatrix}$	`K = np.zeros([3,3])`
	$K = \begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}$	`K = np.eye(3)`
	$K = \begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$	`K = np.ones([3,3])`
	$K = \begin{bmatrix}1&4&7\\2&5&8\\3&6&9\end{bmatrix}$	`K = np.array([[1,4,7],[2,5,8],[3,6,9]])`

Accès à un terme d'une matrice ou d'un vecteur	$K = \begin{bmatrix}1&4&7\\2&5&8\\3&\mathbf{6}&9\end{bmatrix}$
	$F = \begin{bmatrix}0\\\mathbf{0}\\0\end{bmatrix}$; $G = \begin{bmatrix}0\\\mathbf{0}\\0\end{bmatrix}$
		`K[2,1]`
		`F = np.zeros([3])` → `F[1]`
		`G = np.zeros([3,1])` → `G[1,0]` **ou** `G[1,:]`

Copie d'une matrice ou d'un vecteur		`B = np.copy(A)`
Transposition d'une matrice (2 dimensions !)		`K.T` ou `np.transpose(K)`
Produit matriciel Produit scalaire	$\begin{bmatrix}1&4&7\\2&5&8\\3&6&9\end{bmatrix}\begin{bmatrix}1\\2\\3\end{bmatrix} = \begin{bmatrix}30\\36\\42\end{bmatrix}$ $UV = \begin{bmatrix}1\\2\\3\end{bmatrix}\begin{bmatrix}1\\2\\3\end{bmatrix} = 14$	`np.dot(K,F)` ≠ `np.dot(F,K)` `np.dot(U,V)` = `np.dot(V,U)`
Récupération d'une portion d'array **ATTENTION :** **[,] OUI - ~~[][]~~ ~~NON~~**	$K = \begin{bmatrix}1&4&7\\2&5&8\\3&6&9\end{bmatrix}$	Ligne : `K[1,:]` ou `K[1]` Et un exemple qui sert à rien : `K[1][:]`
		Colonne : Attention ! `L[:,1]` OUI - ~~`L[:][1]`~~ NON (ligne 2…)
		`K[0:2,0:2]`
Nb lignes colonnes d'un array de 2 dimensions		`l,c = np.shape(K)`
Dim d'un array à 3 dimensions (par exemple)		`l,c,n = np.shape(K)` `l,c = np.shape(K)` ne fonctionne pas
Nombre de termes d'un array		`np.size(K)`
Produit termes à termes	$U * V = \begin{bmatrix}a\\b\\c\end{bmatrix} * \begin{bmatrix}d\\e\\f\end{bmatrix} \Longrightarrow \begin{bmatrix}ad\\be\\cf\end{bmatrix}$	`U*V`
Inversion d'une matrice		`np.linalg.inv(K)`
Résolution d'un système		`[x,y,z] = np.linalg.solve(K,F)`
Calcul d'un déterminant		`np.linalg.det(K)`

Une résolution

$$\begin{cases} x+2z=1 \\ 3y+4z=2 \\ 5y=3 \end{cases} \quad ; \quad KU=F \quad ; \quad K=\begin{bmatrix} 1 & 0 & 2 \\ 0 & 3 & 4 \\ 0 & 5 & 0 \end{bmatrix} \quad ; \quad U=\begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad ; \quad F=\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Import de Numpy	`import numpy as np`

Création de la matrice K	`K = np.zeros([3,3])`
	`K[0,0] = 1`
	`K[0,2] = 2`
	`K[1,1] = 3`
	`K[1,2] = 4`
	`K[2,1] = 5`

Création du vecteur F	`F = np.array([1,2,3])`
Résolution	`x,y,z = np.linalg.solve(K,F)`

Les slices

Principe	Ligne i	`A[i,:]`
	Colonne i	`A[:,i]`
	Portion carrée entre les lignes et colonnes i et j-1 incluses	`A[i:j,i:j]`
	Modification d'une ou plusieurs valeurs	`A[...,...] = 1`
Avantage	L'utilisation de slices au lieu de doubles boucles for sur une matrice par exemple, change drastiquement le temps d'exécution (ordre de grandeur de 100 à 250 fois plus rapide)	

Remarques

Création d'une matrice de l lignes, c colonnes, et contenant des nuplets	`Mat = np.zeros((l,c,n))`
Test d'égalité	Attention : le test d'égalité renvoie un array rempli de True et False selon l'égalité de chaque composante. Il faudra donc vérifier chaque terme un par un

Transposition	Pour que la transposée fonctionne, il est nécessaire de créer des array à 2 dimensions. Ainsi, initialiser puis remplir :
	`A = np.zeros([3,1])`
	`A[:,0] = [1,2,3]`
	`AT = np.transpose(A)`

Append	N'existe pas sur les array

Avantages

Complexité	Gain en temps et en espace
Opérations	Avec les array, on peut réaliser une somme, une multiplication ou division par un réel, voire même réaliser des opérations, par exemple np.exp(M), np.cos(M), 1/M etc
Récupération dans des listes	On peut récupérer une colonne (ou une portion quelconque de liste) simplement en passant par un array. `L = [[1,2,3],[4,5,6],[7,8,9]]` Méthode liste : `Col1 = [L[i][0] for i in range(len(L))]` Méthode array : `La = np.array(L)` puis `Col1 = La[:,0]` Pour recréer une liste à partir d'un array : `L = [A[i] for i in range(len(A))]`

Types des nombres

Attention à l'overflow. L'array choisi automatiquement les types des nombres qui le composent. Ainsi, un entier est codé en int32 par exemple.
On peut prédéfinir les types : `D = np.array([400000000],dtype='float64')`