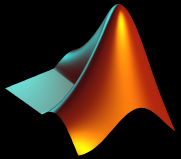
Introduction à Python

Jean-Christophe LOISEAU

Arts & Métiers Institute of Technology, 2021-2022

NumPy : bibliothèque (ou package) pour **Python** destinée à manipuler des **matrices** ou **tableaux multi-dimensionels** ainsi que des fonctions mathématiques opérant sur ces tableaux.


julia

```
    def matvec(A, x):
        m, n = len(A), len(A[0])
        b = [sum(A[i][j]*x[j] for j in range(n)) for i in range(m)]
        return b


    b = matvec(A, x)

```

```
1    import numpy as np
2
3    b = A @ x
4
```

Performances pour une matrice $\boldsymbol{A}$ de taille 10×10.

	Addition	Transposition	Produit matrice-vecteur
Python	$15.2\ \mu s$	$10\ \mu s$	$14.9\ \mu s$
NumPy	$0.5\ \mu s$	$0.1\ \mu s$	$1.2\ \mu s$

Performances pour une matrice $\boldsymbol{A}$ de taille 1000×1000.

	Addition	Transposition	Produit matrice-vecteur
Python	$0.136\ \mathrm{s}$	$0.111\ \mathrm{s}$	$0.1\ \mathrm{s}$
NumPy	$2.5\ \mathrm{ms}$	$0.15\ \mu\mathrm{s}$	$0.5\ \mathrm{ms}$

```python
x = [1, 2]
# [1, 2]

x.append('Je suis du texte')
# [1, 2, 'Je suis du texte']

x.extend([4, 5])
# [1, 2, 'Je suis du texte', 4, 5]

y = ['Encore du texte', 8]
z = x + y
# [1, 2, 'Je suis du texte', 4, 5, 'Encore du texte', 8]

```

```
1     import numpy as np
2
3     x = np.array([1, 2])
4     # array([1, 2])
5
6     y = x + np.array([4, 5])
7     # array([5, 7])
8
9     z = x + np.array([4, 5, 6])
10    # Affiche une erreur !
11
```

```
1    np.array(object, dtype=None, ndim=0)
2
```

- `object` : Typiquement une liste de nombres pour un vecteur, ou une liste de liste de nombres pour une matrice.
- `dtype` : Le type choisi pour le stockage des données (optionnel).

- `np.bool` : Booléen (`True` ou `False`)
- `np.int8` : Entier (-128 à 127)
- `np.int16` : Entier (-2^{15} à $2^{15}-1$)
- `np.int32` : Entier (-2^{31} à $2^{31}-1$)
- `np.int64` : Entier (-2^{63} à $2^{63}-1$)
- `np.uint8` : Entier (0 à 255)
- `np.unint16` : Entier (0 à $2^{16}-1$)
- `np.unint32` : Entier (0 à $2^{32}-1$)
- `np.unint64` : Entier (0 à $2^{64}-1$)
- `np.float16` : Flottant (demi-précision)
- `np.float32` : Flottant (simple précision)
- `np.float64` : Flottant (double précision)
- `np.complex64` : Complexe (simple précision)
- `np.complex128` : Complexe (double précision)

```
x = np.array([1, 2, 3])
x.dtype
# dtype('int64')

x = np.array([1.0, 2, 3])
x.dtype
# dtype('float64')

x = np.array([1, 2, 3], dtype=np.float32)
x.dtype
# dtype('float32')

```

```
x = np.array([1, 2, 3])
x.shape
# (3,)

x = np.array([[1, 2, 3]])
x.shape
# (1, 3)

x = np.array([[1], [2], [3]])
x.shape
# (3, 1)

```

```
A = np.array([[1, 2], [4, 5]])
# array([[1, 2],
#        [3, 4]])

A.dtype
# dtype('int64')

A.shape
# (2, 2)
```

```
A[0, 0]
# 1

A[0] # ou A[0, :]
# array([1, 2])

A[:, 0]
# array([1, 3])
```

```
z = 1 + 1j * 2
# z = 1 + 2i

z = np.array([1.0, 2 + 1j*100])
# z = [1 , 2 + 100i]

z.real
# [1 , 2]

z.imag
# [0 , 100]

z.conj()
# [1 , 2 - 100i]

```

Opérations sur des arrays numpy

- Création d'un array de 0 : `np.zeros((d0, d1, ..., dn), dtype=None)`

```
x = np.zeros(10)
x = np.zeros((4, 3), dtype=np.int32)
x = np.zeros((2, 4, 10), dtype=np.complex128)
```

- Création d'un array de 1 : `np.ones((d0, d1, ..., dn), dtype=None)`

```
x = np.ones(10)
x = np.ones((4, 3), dtype=np.int32)
x = np.ones((2, 4, 10), dtype=np.complex128)
```

```python
x = np.array([1, 2, 3])

x.sum()      # np.sum(x)
# 6

x.prod()     # np.prod(x)
# 6

x.mean()     # np.mean(x)
# 2

x.std()      # np.std(x)
# 0.81649
```

```python
x.max()     # np.max(x)
# 3

x.min()     # np.min(x)
# 1

x.argmax() # np.argmax(x)
# 2

x.argmin() # np.argmin(x)
# 0

x.abs() # np.abs(x)
```

```python
A = np.array([[1, 2], [3, 4]])
# array([[1, 2],
#        [3, 4]])

A.sum()            # np.sum(A)
# 10

A.sum(axis=0)      # np.sum(A, axis=0)
# array([4, 6])

A.sum(axis=1)      # np.sum(A, axis=1)
# array([3, 7])

A.T                # np.transpose(A)
# array([[1, 3],
#        [2, 4]])
```

```
y = np.exp(x)

y = np.cos(x)

y = np.sin(x)

y = np.tan(x)

y = np.cosh(x)

y = np.sinh(x)
```

```
y = np.log(x)

y = np.arccos(x)

y = np.arcsin(x)

y = np.arctan(x)

y = np.arccosh(x)

y = np.arcsinh(x)
```

- **np.linalg** : Méthodes de base pour l'algèbre linéaire (e.g. inversion de matrice, résolution de système linéaire, valeurs propres, etc).

- **np.random** : Génération de nombres aléatoires provenant de différentes distributions (e.g. normale, Laplace, Gamma, etc).

- **np.fft** : Transformées de Fourier rapides (1D, 2D et nD) utiles pour le traitement du signal et des images.

```
1    import numpy as np
2    import numpy.linalg as npl
3
```

```
x, y = np.array([1, 2]), np.array([3, 4])

# Produit scalaire.
c = np.vdot(x, y)

# Norme Euclidienne.
c = npl.norm(x)

```

```
A = np.array([[1, 2], [3, 4]])

# Inverse.
B = npl.inv(A)

# Determinant.
d = npl.det(A)

# Valeurs propres et vecteurs propres.
vals, vecs = npl.eig(A)      # vals = npl.eigvals(A)

# Elevation d'une matrice A a la puissance k .
B = npl.matrix_power(A, k)

```

```
    # Produit matrice-matrice / matrice-vecteur.
    y = A @ x      # np.dot(A, x)      A.dot(x)      npl.matmul(A, x)

    # Resolution de systeme lineaire.
    x = npl.solve(A, b)

    # Resolution au sens des moindres-carres.
    x = npl.lstsq(A, b)

```

Résoudre $\boldsymbol{Ax} = \boldsymbol{b}$ numériquement.

```
x = npl.inv(A) @ b
```

ou

```
x = npl.solve(A, b)
```

```
1    import numpy as np
2    import numpy.random as npr
3
```

```
    # Genere n nombres aleatoires issus d'une distribution normale.
    x = npr.normal(loc=0.0, size=1.0, size=n)

    # Genere n nombres aleatoires issus d'une distribution uniforme.
    x = npr.uniform(low=0.0, high=1.0, size=n)


    import matplotlib.pyplot as plt
    plt.hist(x)

```

```
1    # Sauvegarde un array en format texte.
2    np.savetxt('mon_fichier.dat', x)
3
4    # Charge un array a partir d'un fichier texte.
5    x = np.loadtxt('mon_fichier.dat')
6
```

Pour en savoir plus, rendez-vous sur `https://numpy.org/`