21

Calcul matriciel

Remarque. Le calcul matriciel avec SCILAB a été vu précédemment, de même que la manipulation de tableaux avec NUMPY. On s'intéresse ici à la réalisation des opérations matricielles avec NUMPY. □

I. Calcul matriciel avec Numpy/Scipy

`A+c`	Ajoute `c:float` à tous les coefficients de `A:array(n,p)`.
`A+B`	Somme des deux tableaux, `A:array(n,p)` et `B:array(n,p)`.
`A*B`	Produit, coefficient par coefficient, de `A:array(n,p)` et `B:array(n,p)`.
`A**k`	Tous les coefficients de `A:array(n,p)` sont élevés à la puissance `k:float`.
`np.dot(A,B)`	Produit matriciel de `A:array(n,p)` par `B:array(p,m)`.
`np.dot(A,x)`	Produit matriciel de `A:array(n,p)` par `x:array(p)`.
`np.hstack(A,B)`	Juxtapose horizontalement `A:array(n,p)` et `B:array(n,m)`.
`np.vstack(A,B)`	Juxtapose verticalement `A:array(n,p)` et `B:array(m,p)`.
`np.linalg.matrix_power(A,k)`	Puissance matricielle, `A:array(n,n)` et `k:int`.
`np.linalg.det(A)`	Déterminant de `A:array(n,n)`
`np.linalg.inv(A)`	Matrice inverse de `A:array(n,n)`.
`A.transpose()`	Transposée de `A:array(n,p)`.
`np.eye(n)`	Matrice identité de taille `n:int`.
`np.zeros((n,p))`	Matrice nulle à `n` lignes et `p` colonnes.
`np.zeros(A.shape)`	Matrice nulle de même taille que `A`.

II. Résolution de systèmes linéaires

Exemples.

- $(S_1) : \begin{cases} x+y=5 \\ x+2y=3 \end{cases}$ (unique solution), noté matriciellement $A\begin{bmatrix} x \\ y \end{bmatrix} = u$;
- $(S_2) : \begin{cases} x+y=5 \\ 2x+2y=10 \end{cases}$ (infinité de solutions), noté matriciellement $B\begin{bmatrix} x \\ y \end{bmatrix} = v$;
- $(S_3) : \begin{cases} x+y=5 \\ 2x+2y=0 \end{cases}$ (pas de solution), noté matriciellement $B\begin{bmatrix} x \\ y \end{bmatrix} = w$. □

```
x0=linsolve(A,-u)
 x0  =
    7.
  - 2.
```

⚠ Avec SCILAB, **`linsolve(A,b)`** résout le système de $Ax+b=0$. Pour résoudre $Ax=b$, on utilise donc **`linsolve(A,-b)`**.

```
x0=linsolve(B,-v)
 x0  =
    2.5
    2.5
[x0,ker]=linsolve(B,-v)
 ker  =
    0.7071068
  - 0.7071068
 x0  =
    2.5
    2.5
```

```
linsolve(B,-w)
AVERTISSEMENT : Contraintes linéaires en conflit.
 ans  =
     []
```

```
lsq(B,w)
 ans  =
    0.5
    0.5
```

◇ **`lsq(B,w)`** détermine un vecteur $\begin{bmatrix} x \\ y \end{bmatrix}$ tel que $B\begin{bmatrix} x \\ y \end{bmatrix} - w$ soit « le plus petit possible. »

```
print np.linalg.solve(A,u)
```

```
[ 7. -2.]
```

⚠ Avec PYTHON, **`solve(A,b)`** résout $Ax=b$ mais la matrice A doit être inversible.

```
np.linalg.solve(B,v) # Ne marche pas : B non inversible
np.linalg.solve(B,w) # Idem
```

```
print np.linalg.lstsq(B,v)[0] # Comme lsq avec Scilab
```

```
[ 2.5 2.5]
```

```
print np.linalg.lstsq(B,w)[0] # Idem
```

```
[ 0.5 0.5]
```