

# Calcul Scientifique

## Cours 2: Opération de base d'algèbre linéaire

Alexis Lechervy



# Sommaire

- 1 Introduction à BLAS
- 2 Calculs vecteurs/vecteurs
- 3 Calculs matrices/vecteurs
- 4 Calculs matrices/matrices

# Problématique

## Enjeux

Dans de nombreux problèmes scientifiques, **certaines opérations sont récurrentes**. (par exemple : une étude<sup>1</sup> de 1964 sur le supercalculateur IBM 7030 montre que 11% des calculs les plus fréquents sur les nombres à virgule flottante représentent la sommation simple.)  $\implies$  il devient important d'**optimiser ces opérations** (exemple : le circuit shifter de l'IBM 7030 pour la sommation en virgule flottante).

## Exemple : le produit matricielle $C = AB$

Approche naïve :

- $\forall i, j \ c_{ij} = \sum_{k=1}^n a_{i,k} b_{k,j}$
- Nécessite trois boucles (sur les  $i, j$  et  $k$ ).
- Pas du tout optimal en calcul, en accès mémoire, n'utilise pas bien les caches processeurs...
- Une implémentation plus performante est plus longue et nécessite une bonne connaissance des optimisations matérielles possibles.

# BLAS : Basic Linear Algebra Subprograms

## BLAS : Basic Linear Algebra Subprograms

**BLAS** est une **interface de programmation** réalisant des opérations de bases d'**algèbre linéaire** (addition de vecteurs, produit scalaire, norme de vecteur, multiplication matricielle ...). Ces fonctions sont largement utilisées pour du **calcul haut performance** et ont été largement optimisées par les constructeurs matériel comme Intel, AMD, Nvidia.

## Historique

- Avant 1970, les codes utilisant du calcul rapide utilisaient des **fonctions optimisées écrites directement en langage machine**. => ces fonctions étaient généralement pas portable et le code les utilisant devaient être mis à jours à chaque évolution.
- Entre 1972-1978, à l'initiative de l'University of Tennessee, les fonctions les plus utilisés ont été identifiées et des sous-routines en Fortran 77 à nom génériques ont été créées. Les fonctions ont alors pu être utilisées en mode "boite noire" tout en s'améliorant en permanence.
- En 1979, publication de BLAS niveau 1 et première implémentation dans des bibliothèques open source comme LAPACK.
- En 1984-1986, l'évolution matérielle comme l'apparition des co-processeurs, permettant d'effectuer des opérations (somme, produit...) en même temps, conduit à la publication de nouvelles fonctions : BLAS niveau 2.
- En 1987-1988, l'utilisation de mémoire cache à accès rapide, les progrès du calcul parallèle, on conduit les programmeurs à privilégier les calculs par blocs : BLAS niveau 3.

# BLAS, trois niveaux de fonctions

## BLAS, niveau 1

Regroupe les opérations sur les vecteurs et entre vecteurs dont le résultat est une valeur ou un vecteur [http://www.netlib.org/blas/#\\_level\\_1](http://www.netlib.org/blas/#_level_1).

### Exemples

- Les opérations de la forme :  $\alpha x + y$ , avec  $x, y$  des vecteurs et  $\alpha$  un scalaire,
- Les produits scalaires :  $\langle x, y \rangle = \sum_i x_i y_i$ ,
- La norme euclidienne d'un vecteur :  $\|x\|^2 = \sum_i x_i^2$ ,
- La norme 1 d'un vecteur :  $\|x\|_1 = \sum_i |x_i|$ ,
- ...

# BLAS, trois niveaux de fonctions

## BLAS, niveau 2

Regroupe les opérations entre un vecteur et une matrice dont le résultat est un vecteur ou une matrice [http://www.netlib.org/blas/#\\_level\\_2](http://www.netlib.org/blas/#_level_2).

## Exemples

- Les opérations de la forme :  $\alpha Ax + \beta y$ , avec  $x, y$  des vecteurs,  $\alpha, \beta$  des scalaires et  $A$  une matrice,
- Résolution de l'équation  $Tx = y$  avec  $x, y$  des vecteur et  $T$  une matrice triangulaire ( $x$  est l'inconnu retourné par la fonction),
- Les opérations de la forme  $\alpha xy^T$ , avec  $x, y$  des vecteurs et  $\alpha$  un scalaire,
- ...

# BLAS, trois niveaux de fonctions

## BLAS, niveau 3

Regroupe les opérations entre matrices dont le résultat est une matrice.

[http://www.netlib.org/blas/#\\_level\\_3](http://www.netlib.org/blas/#_level_3)

## Exemples

- Les opérations de la forme :  $\alpha AB + \beta C$ , avec  $\alpha, \beta$  des scalaires et  $A, B, C$  des matrices,
- Résolution de l'équation  $TX = \alpha B$ , avec  $\alpha$  un scalaire,  $T$  est une matrice triangulaire et  $B, X$  des matrices ( $X$  est l'inconnu retourné par la fonction),
- ...

# BLAS en pratique

## Info

L'ensemble des fonctions de la norme est résumé ici :

<http://www.netlib.org/blas/blasqr.pdf>

## Implémentations de BLAS

Il existe de nombreuses implémentations de BLAS commerciale et open source :

- ATLAS,
- Intel MKL,
- ACML (AMD Core Math Library),
- Accelerate (Apple framework for MacOS),
- cuBLAS (Optimized BLAS for NVIDIA ),
- clBLAS (OpenCL implementation of BLAS),
- OpenBLAS,
- GotoBLAS,
- EigenBLAS...



# BLAS et python

## BLAS dans numpy et scipy

Les bibliothèques python **numpy** et **scipy** s'appuient sur une implémentation BLAS pour fonctionner. Même s'il est possible d'appeler directement ces fonctions (<https://docs.scipy.org/doc/scipy/reference/linalg.blas.html>), on les **utilise généralement indirectement**.

## Remarques d'optimisation

Lors d'une **installation basique** de numpy et scipy une **version non optimisée de BLAS** est utilisée. Pour optimiser vos temps de calculs, vous devez **installer une implémentations indépendante de BLAS** et la relier à votre installation de numpy et scipy.

<https://docs.scipy.org/doc/numpy-1.10.1/user/install.html>

# Sommaire

- 1 Introduction à BLAS
- 2 Calculs vecteurs/vecteurs
  - Présentation des calculs vecteurs/vecteurs
  - Exemples d'utilisation
- 3 Calculs matrices/vecteurs
- 4 Calculs matrices/matrices

# Calculs vecteurs / vecteurs

## Combinaison linéaire de vecteur

Si  $a$  est un nombre et  $v_1, v_2$  des array numpy à  $d$  dimension, on peut écrire :

$$v = a * v1 + v2$$

Rq :  $v1$  et  $v2$  doivent avoir les même dimensions.

## Produit terme à terme entre deux vecteurs $v_1$ et $v_2$

$$v = v1 * v2$$

## Produit scalaire entre deux vecteurs $v_1$ et $v_2$

*Rappel mathématique* : Le produit scalaire correspond à la somme des produits des valeurs des deux vecteurs :

$$\langle v_1, v_2 \rangle = \sum_i v_{1i} v_{2i}.$$

*Implémentation numpy* :

```
p = v1.dot(v2)
```

```
p = v2.dot(v1)
```

```
p = np.dot(v1, v2)
```

# Remarque sur le produit scalaire

## Le produit scalaire

Le produit scalaire correspond à la formule :

$$\langle v_1, v_2 \rangle = \sum_i v_{1i} v_{2i}.$$

On peut l'implémenter avec la fonction *dot* de numpy.

Autre implémentation possible du Produit scalaire entre deux vecteurs  $v_1$  et  $v_2$

```
p = np.sum(v1 * v2)
```

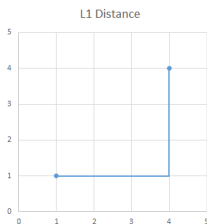
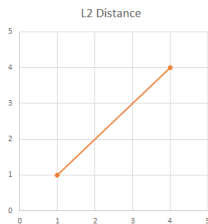
# Norme d'un vecteur

## Définition simplifié

La norme est l'extension de la notion de valeurs absolue aux vecteurs. Elle représente la «distance» entre le point 0 et le point à l'extrémité du vecteur mesuré.

## La notion de distance

Il existe plusieurs façon de définir une distance entre deux pointst.



## Norme euclidienne d'un vecteur $v$

Formule de math :  $\|v\| = \sqrt{\sum_i x_i^2}$

Implémentation :

```
n2 = np.linalg.norm(v)
```

```
n2 = np.linalg.norm(v,2)
```

```
n2 = np.sqrt(np.sum( v **2))
```

## Norme L1 d'un vecteur

Formule de math :  $|v|_1 = \sum_i |x_i|$

Implémentation :

```
n1 = np.linalg.norm(v,1)
```

```
n1 = np.sum( np.abs(v))
```

# Cas de la somme d'un scalaire et d'un vecteur

## Somme d'un scalaire $a$ et d'un vecteur $v$

Sommer un nombre et un vecteur revient au même que de sommer le vecteur avec un vecteur de même taille où le nombre aurait été recopié sur toutes les dimensions.

## Exemple

Les instructions suivantes sont quasi-équivalente en terme de résultat :

$$\begin{aligned} &v + a \\ &v + a * \text{np.ones}(v.\text{shape}) \end{aligned}$$

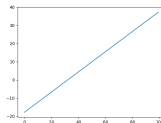
# Exemples d'utilisation

## Conversion d'un vecteur de température en Fahrenheit en Celsius

Il est possible de transformer une température en Fahrenheit en Celsius grâce à la formule  $t_{\text{celsius}} = \frac{5}{9} t_{\text{Fahrenheit}} - \frac{160}{9}$ .

### Code correspondant

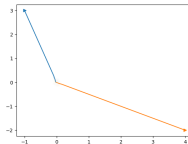
```
f = np.arange(0,100) # Fahrenheit entre 0 et 100  
c = 5/9 * f - 160/9 # conversion en degree celsius  
plt.plot(f,c) # affichage de c en fonction de f  
plt.show()
```



# Calcul d'angle entre deux vecteurs

## Déclaration des vecteurs

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



## Calcul du produit scalaire et des normes des vecteurs

```
p = np.dot(x,y) # produit scalaire entre x et y ( $\langle x, y \rangle$ )  
n_x = np.linalg.norm(x) # norme de x ( $\|x\|$ )  
n_y = np.linalg.norm(y) # norme de y ( $\|y\|$ )
```

## Calcul de l'angle entre les deux vecteurs

Rappel sur le cosinus :  $\cos(\vec{x}, \vec{y}) := \frac{\langle \vec{x}, \vec{y} \rangle}{\|x\| \|y\|}$

Calcul de l'angle en degré : `np.arccos(p/(n_x * n_y)) * 180/np.pi`



# Sommaire

- 1 Introduction à BLAS
- 2 Calculs vecteurs/vecteurs
- 3 Calculs matrices/vecteurs
  - Présentation des calculs matrices/vecteurs
  - Exemples d'application
  - Cas particuliers d'application vecteur/vecteur  $\rightarrow$  Matrice
- 4 Calculs matrices/matrices

# Calculs matrice / vecteur

## Combinaison linéaire de type $\alpha Ax + \beta y$

$$a * A.dot(x) + b*y$$

Attention  $A$  est une matrice mais  $x$  et  $y$  sont des vecteurs soit des **array numpy de dimension (d,)** et non (d,1) ou (1,d).

## Exemple : Évaluation rapide d'un ensemble d'équation

Problème :

$$\begin{cases} 2x + 3y + 1 = ? \\ -4x + 12z = ? \\ 23x + 4y - 42z - 4 = ? \\ 17z - 2 = ? \end{cases} \quad \text{pour } x = -11, y = 13 \text{ et } z = 16 \quad (1)$$

Réécriture

$$A = \begin{bmatrix} 2 & 3 & 0 \\ -4 & 0 & 12 \\ 23 & 4 & 42 \\ 0 & 0 & 17 \end{bmatrix}, b = \begin{bmatrix} 1 \\ 0 \\ -4 \\ -2 \end{bmatrix}, x = \begin{bmatrix} -11 \\ 13 \\ 16 \end{bmatrix}, Ax + b = ? \quad (2)$$

Solution numpy :  $A.dot(x)+b$

# Vecteur mathématique et array numpy

## Remarque sur les array numpy

Un vecteur mathématique peut être représenté par différent array numpy ayant des comportements différents.

```
v1 = np.array([1,2]) \# vecteur generique
```

```
v1.shape
```

→ (2,)

```
v2 = np.array([[1,2]]) \#vecteur ligne(1 ligne|2 colonnes)
```

```
v2.shape
```

→ (1,2)

```
v3 = np.array([[1],[2]]) \#vecteur colonne(2 lignes|1 colonne)
```

```
v3.shape
```

→ (2,1)

# Transposée

## Mathématiquement

La matrice transposée est obtenue en échangeant les lignes et les colonnes.

$$\begin{bmatrix} 1 & 11 \\ 2 & 22 \\ 3 & 33 \end{bmatrix}^T = \begin{bmatrix} 1 & 2 & 3 \\ 11 & 22 & 33 \end{bmatrix} \quad (3)$$

## Transposé de matrice

```
M = np.array([[1,11],[2,22],[3,33]])
```

```
Mt = M.T
```

```
    -> array([[ 1,  2,  3],
             [11, 22, 33]])
```

```
M.shape
```

```
    ->      (3,2)
```

```
Mt.shape
```

```
    ->      (2,3)
```

# Transposée de vecteur

## Définition d'un vecteur $v1$

Mathématiquement :

$$v1 = [1, 2] \quad (4)$$

Avec numpy : `v1 = np.array([1,2])`

## Transposé sur $v1$

```
v1t = v1.T
```

```
v1t
```

```
→ array([1, 2])
```

```
v1t.shape
```

```
→ (2,)
```

L'opérateur de transposé n'a pas d'effet sur le vecteur  $v1$ .

## Définition d'un vecteur $v_2$

Mathématiquement :

$$v_2 = [1, 2] \quad (5)$$

Avec numpy : `v2 = np.array([[1,2]])`

## Transposé sur $v_2$

`v2t = v2.T`

`v2t`

→ `array([[1],  
[2]])`

`v2t.shape`

→ `(2,1)`

L'opérateur de transposé s'est appliqué en transformant un vecteur ligne en vecteur colonne.

## Définition d'un vecteur $v3$

Mathématiquement :

$$v3 = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \quad (6)$$

Avec numpy : `v3=np.array([[1],[2]])`

## Transposé sur $v3$

`v3t = v3.T`

`v3t`

→ `array([[1, 2]])`

`v3t.shape`

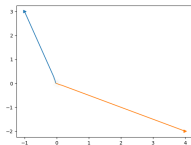
→ `(1, 2)`

L'opérateur de transposé s'est appliqué en transformant un vecteur colonne en vecteur ligne.

# Calcul d'angle entre deux vecteurs (une autre solution)

## Déclaration des vecteurs

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



## Calcul du produit scalaire et des normes des vecteurs

```
p = x.T.dot(y)  
n_x = np.sqrt(x.T.dot(x))  
n_y = np.sqrt(y.T.dot(y))
```

## Calcul de l'angle entre les deux vecteurs

Rappel sur le cosinus :  $\cos(\vec{x}, \vec{y}) := \frac{\langle \vec{x}, \vec{y} \rangle}{\|\vec{x}\| \|\vec{y}\|}$

Calcul de l'angle en degré :  $\text{np.arccos}(p/(n\_x * n\_y)) * 180/\text{np.pi}$

**Attention : Le résultat n'est plus une valeur, mais un tableau de dimensions (1,1).**



# Les opérations de la forme $\alpha xy^T$

## Résultat mathématique attendu

$$\alpha = 1, x = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \text{ et } y = \begin{bmatrix} 11 \\ 12 \end{bmatrix} \Rightarrow \alpha xy^T = \begin{bmatrix} 11 & 12 \\ 22 & 24 \end{bmatrix}$$

## Implémentation correct

```
x = np.array([[1],[2]])  
y = np.array([[11],[12]])  
x.dot(y.T)  
    -> array([[11, 12],  
             [22, 24]])
```

## Remarque avec des vecteurs numpy (2,)

```
x = np.array([1,2])  
y = np.array([11,12])  
x.dot(y.T)  
    -> 35
```

Ne calcul pas la matrice attendu mais le produit scalaire entre les deux vecteurs.

# Trouver la ville la plus proche à vol d'oiseau

Calcul de la distance la plus petit entre un point et un ensemble de point

- Les coordonnées des villes seront stockés ligne par ligne dans une matrice  $M$ .
- La coordonnée de l'utilisateur (point rouge) sera représenté par un vecteur  $v$ .



# Distances

Distances entre deux points  $x$  et  $y$

Mathématiquement :

$$d(x, y) = \|x - y\| \quad (7)$$

$$= \sqrt{\sum_i (x_i - y_i)^2} \quad (8)$$

En numpy :

```
np.sqrt(np.sum((x-y)**2))
```

Distances entre un point  $v$  et tout les points  $M$  d'un ensemble (solution "naïve")

```
d=np.zeros(n)
for i in range(n) :
    d[i] = np.sqrt(np.sum((M[i]-v)**2))
```

Problème : Les boucles for ne sont pas performante et doivent être évitées !!

# Distance

## Distances entre deux points $x$ et $y$

Mathématiquement :

$$d(x, y) = \|x - y\| \quad (9)$$

$$= \sqrt{\sum_i (x_i - y_i)^2} \quad (10)$$

$$= \sqrt{\sum_i x_i^2 - 2x_i y_i + y_i^2} \quad (11)$$

$$= \sqrt{\sum_i x_i^2 - 2 \sum_i x_i y_i + \sum_i y_i^2} \quad (12)$$

$$\quad (13)$$

$$= \sqrt{\|x\|^2 - 2\langle x, y \rangle + \|y\|^2} \quad (14)$$

$$\quad (15)$$

En numpy :

```
np.sqrt(np.sum(x**2) - 2*x.T.dot(y) + np.sum(y**2) )
```

# Distance

Distances entre deux points  $x$  et  $y$

Mathématiquement :

$$d(x, y) = \sqrt{\|x\|^2 - 2\langle x, y \rangle + \|y\|^2} \quad (16)$$

(17)

En numpy :

$$\text{np.sqrt}(\text{np.sum}(x^{**2}) - 2 * x.T.\text{dot}(y) + \text{np.sum}(y^{**2}))$$

Distances entre un point  $v$  et tout les points  $M$  d'un ensemble (bonne solution)

$$d = \text{np.sqrt}(\text{np.sum}(M^{**2}, \text{axis}=1) - 2 * M.\text{dot}(v) + \text{np.sum}(v^{**2}))$$

Distances entre un point  $v$  et un ensemble de points  $M$

$$\text{np.min}(d)$$

# Sommaire

- 1 Introduction à BLAS
- 2 Calculs vecteurs/vecteurs
- 3 Calculs matrices/vecteurs
- 4 Calculs matrices/matrices
  - Présentation des opérations matrice/matrice
  - Exemple d'application

# Le produit matriciel de Hadamard

## Définition mathématique

Le produit de Hadamard  $M$  des matrices  $A$  et  $B$  est le produit terme à terme des deux matrices.

$$M_{ij} = A_{ij}B_{ij}$$

## En numpy

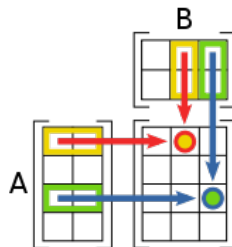
$$M = A * B$$

# Le produit matricielle

## Définition mathématique

Le produit matricielle  $M$  des matrice  $A$  et  $B$  est :

$$M_{ij} = \sum_k A_{ik} B_{kj}$$



## En numpy

$$M = A.dot(B)$$

ou

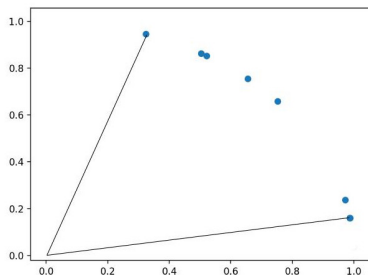
$$M = np.dot(A, B)$$



# Angle d'un cône englobant

## Problématique

Calculer l'angle max entre un groupe de vecteur de norme 1.



## Solution

```
angle = np.arccos(x.dot(x.T) ) * 180/np.pi  
print('Solution : ',np.max(angle))
```