Méthodes numériques en langage Python

Algèbre linéaire

R. Flamary

12 septembre 2019

Plan du cours

Algebre linéaire

Principe

- Outils mathématiques permettant de modéliser les espaces vectoriels et des relations linéaires entre vecteurs.
- Relations linéaires simples à interpréter.
- Modélisation des données numériques sous la forme de tableau.

Exemples d'utilisation

- Systèmes de recommandation (Netflix, Amazon, Criteo).
- Moteurs de recherche (Google, Bing, Duckduckgo).
- Prédiction de valeurs réelles (Météo, cours de la bourse).
- Traitement du signal et des images (systèmes linaires, filtrage, reconstruction).
- Simulations physiques (méthodes d'élément finis, équations de la chaleur).

Espaces de dimension finie

Espace vectoriel

Un espace vectoriel sur K est est un ensemble E muni de deux lois :

- L'addition $+ : E^2 \to E$, opération commutative et associative d'élément neutre $\mathbf{0}$ appelé vecteur nul.
- La multiplication par un scalaire $\cdot : K \times E \to E$, opération distributive et d'élément neutre 1.

Un vecteur de $\mathbf{v} \in E$ est définit par un symbole en gras dans ce cours.

Famille de vecteurs

- Soit une famille de vecteur $\{\mathbf{v}_i\}_{i=1,\dots,m}$.
- La famille est dite linéairement indépendante si la seule combinaison linéaire de la forme $\sum_i \lambda_i \mathbf{v}_i = 0$ est celle avec $\lambda_i = 0, \quad \forall i$.
- Le sous-espace vectoriel engendré par une famille $\{\mathbf{v}_i\}_{i=1,\dots,m}$ dénoté $Vec(\{\mathbf{v}_i\}_{i=1,\dots,m})$ est l'ensemble des combinaisons linéaires de $\mathbf{v}_i$.
- La famille $\{\mathbf{v}_i\}_{i=1,\dots,m}$ linéairement indépendante est une base de E si $Vec(\{\mathbf{v}_i\}_{i=1,\dots,m}) = E$.

Vecteurs dans $\mathbb{R}^n$

Notation

Les vecteurs de $\mathbb{R}^n$ peuvent être exprimés sous la forme :

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \qquad \text{ou} \qquad \mathbf{v}^\top = [v_1, v_2, \dots, v_n]$$

En l'absence du symbole $\top$, tous les vecteurs sont supposés vecteurs colonne.

Base canonique et norme

- La base canonique de $\mathbb{R}^n$ est la famille de vecteurs $\{\mathbf{v}_i\}_{i=1,\dots,n}$ telle que $v_i = \delta_i$ où δ_i est le vecteur dirac nul sur toutes ses composante à par la composante i égale à 1.
- La norme d'un vecteur de $\mathbb{R}^n$ est définie par

$$\|\mathbf{v}\|^2 = \sum_i v_i^2$$

- La base canonique est orthonormale, c'est à dire orthogonale et chaque élément a une norme unité.

Produit scalaire

- Le produit scalaire entre deux vecteurs $\mathbf{x}$ et $\mathbf{y}$ de $\mathbb{R}^n$ est l'application bilinéaire :

$$\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^\top \mathbf{y} = \sum_i x_i y_i$$

- La norme euclidienne au carré d'un vecteur est son produit scalaire avec lui même :

$$\|\mathbf{v}\|^2 = \langle \mathbf{v}, \mathbf{v} \rangle = \mathbf{v}^\top \mathbf{v}$$

- Un espace vectoriel associé à un produit scalaire est appelé un espace pré-hilbertien.
- Inégalité de Cauchy-Schwartz :

$$\forall \mathbf{x}, \mathbf{y} \in E \quad |\langle \mathbf{x}\,,\, \mathbf{y} \rangle| \;\leq\; \|\mathbf{x}\| \|\mathbf{y}\|$$

- L'angle θ entre deux vecteurs peut être déterminé grâce à

$$\langle \mathbf{x}, \mathbf{y} \rangle = \|\mathbf{x}\| \|\mathbf{y}\| \cos\theta$$

- Le produit scalaire peut être utilisé pour projeter un vecteur sur un autre vecteur.

Application linéaire

Définition

Soient E et F deux espaces vectoriels. Une application f de E vers F est dite linéaire si elle est additive et commute à la multiplication par les scalaires :

$$\begin{aligned} \forall \mathbf{x}, \mathbf{y} \in E, \quad f(\mathbf{x} + \mathbf{y}) &= f(\mathbf{x}) + f(\mathbf{y}), \\ \forall \mathbf{x} \in E, \forall \lambda \in K, \quad f(\lambda \mathbf{x}) &= \lambda f(\mathbf{x}). \end{aligned}$$

Autrement dit, f préserve les combinaisons linéaires.

Noyaux et image

Pour toute application linéaire f de E dans F,

- Les vecteurs $\mathbf{x}$ de E tels que $f(\mathbf{x}) = \mathbf{0}$ forment un sous-espace vectoriel de E, appelé le noyau de f et noté $Ker(f)$.
- les vecteurs $f(\mathbf{x})$ pour $\mathbf{x}$ dans E forment un sous-espace vectoriel de F, appelé l'image de f et noté $Im(f)$.
- Les dimensions des sous-espaces sont liées par le téorème du rang :

$$\dim(E) = \dim\ \mathrm{Ker}(f) + \dim\ \mathrm{Im}(f)$$

Matrice

Notation

Les matrices de $\mathbb{R}^{m \times n}$ sont définies par :

$$\mathbf{A} = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix}, \quad a_{i,j} \in \mathbb{R}, \forall i, j$$

On pourra également utiliser $\mathbf{A}(:, k)$ pour désigner la $k$ième colonne de $\mathbf{A}$ et $\mathbf{A}(k, :)$ pour désigner la $k$ième ligne de $\mathbf{A}$.
Attention, en C on utilise l'indexage à 0, $a_{1,1}$ correspondra donc à l'index entier $(0, 0)$.

Opérations de base

- Transposition

$$\mathbf{C} = \mathbf{A}^\top \quad \Leftrightarrow \quad c_{i,j} = a_{j,i}$$

- Addition

$$\mathbf{C} = \mathbf{A} + \mathbf{B} \quad \Leftrightarrow \quad c_{i,j} = a_{i,j} + b_{i,j}$$

- Multiplication par un scalaire α

$$\mathbf{C} = \alpha \mathbf{A} \quad \Leftrightarrow \quad c_{ij} = \alpha a_{ij}$$

Multiplication matricielle

Multiplication matricielle

Soient deux matrices $\mathbf{A} \in \mathbb{R}^{m \times p}$ et $\mathbf{B} \in \mathbb{R}^{p \times n}$

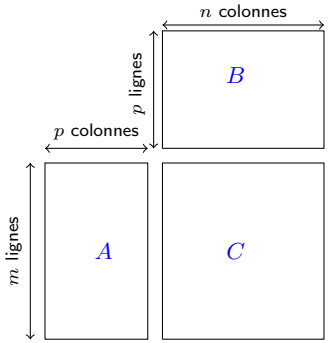
$$\mathbf{C} = \mathbf{AB} \quad \Leftrightarrow \quad c_{i,j} = \sum_{k=1}^{p} a_{i,k} b_{k,j}$$

avec $\mathbf{C} \in \mathbb{R}^{m \times n}$
Attention le produit matriciel n'est pas commutatif ($\mathbf{AB} \neq \mathbf{BA}$).

Multiplication matrice vecteur

Si $\mathbf{B} = \mathbf{b}$ est un vecteur de taille p alors

$$\mathbf{c} = \mathbf{Ab} \quad \Leftrightarrow \quad c_i = \sum_{k=1}^{p} a_{i,k} b_k$$



Conseil : faire le dessin.

Matrice et application linéaire

- Toute application linéaire entre deux espaces munis chacun d'une base finie est représentable par une matrice.
- Une application linéaire $f : \mathbb{R}^n \to \mathbb{R}^m$ peut donc être mise sous la forme :

$$f(\mathbf{x}) = \mathbf{Ax}$$

  - L'image $Im(f)$ de f est le sous espace $Vec(\mathbf{A}(:,k)_{k=1,\ldots,n})$ engendré par les colonnes de $\mathbf{A}$.
  - Le noyau $Ker(f)$ de f est le sous espace orthogonal aux lignes $\mathbf{A}(k,:)$ de $\mathbf{A}$.
- Cas particulier où $\mathbf{A} = \mathbf{v}^\top$ est un vecteur transposé, $f : \mathbb{R}^n \to \mathbb{R}$ est une fonction à valeur réelle.

$$f(\mathbf{x}) = \mathbf{v}^\top \mathbf{x} = \langle \mathbf{v}, \mathbf{x} \rangle = \sum_k v_k x_k$$

  La valeur de la fonction est une somme des composante x_k de $\mathbf{x}$ pondérées par $\mathbf{v}$.
- Cas particulier où $m = n$ et les lignes de $\mathbf{A}$ définissent une base orthonormale de $\mathbb{R}^n$: l'application linéaire est un changement de base.

Matrices particulières

Matrice Diagonale ($m = n$)

$$\begin{bmatrix} a_{1,1} & 0 & \cdots & 0 \\ 0 & a_{2,2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{m,n} \end{bmatrix}$$

- Matrice diagonale :

$$i \neq j \Rightarrow a_{ij} = 0, \quad \forall i, j$$

- Matrice scalaire si $a_{ii} = C, \forall i$.
- Matrice identité si $a_{ii} = 1, \forall i$.

Matrice triangulaire

$$\begin{bmatrix} a_{1,1} & 0 & \cdots & 0 \\ a_{2,1} & a_{2,2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix}$$

- Matrice triangulaire inférieure :

$$i < j \Rightarrow a_{ij} = 0, \quad \forall i, j$$

- $\mathbf{A}^\top$ est triangulaire supérieure.
- Strictement triangulaire inférieure si

$$i \leq j \Rightarrow a_{ij} = 0, \quad \forall i, j$$

Exercice 1 : Multiplication matricielle

Soit les vecteurs et matrice suivants :

$$\mathbf{u} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, \quad \mathbf{v} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \quad \mathbf{A} = \begin{bmatrix} 1 & -1 & 0 \\ -1 & 1 & -1 \\ 0 & -1 & 1 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

Calculer les produits suivants :

$$\mathbf{u}^\top \mathbf{u} = \qquad \mathbf{u}^\top \mathbf{v} = \qquad \mathbf{u}^\top \mathbf{A} \mathbf{v} =$$

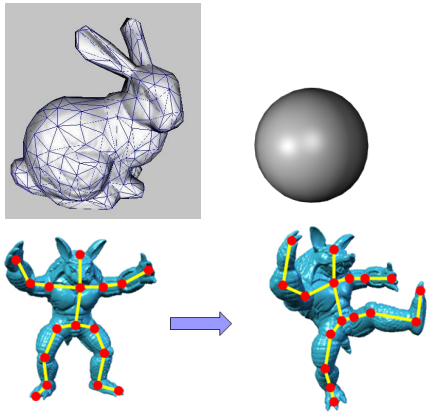
$$\mathbf{u}\mathbf{u}^\top = \qquad \mathbf{u}\mathbf{v}^\top = \qquad \mathbf{A}\mathbf{A} =$$

$$\mathbf{A}\mathbf{u} = \qquad \mathbf{A}\mathbf{v} = \qquad \mathbf{B}^\top \mathbf{u} = \qquad \mathbf{A}\mathbf{B} =$$

Applications de l'algèbre linéaire

Infographie

- L'immense majorité des modèles 3D utilisés dans les jeux vidéo sont représentés par
  - Vecteurs de position dans $\mathbb{R}^3$.
  - Matrice d'adjacence.
- Les algorithmes d'illumination font souvent appel au produit scalaire entre un vecteur normal à la surface et la direction d'une source lumineuse ponctuelle.
- Les rotations et déplacements des membres des models 3D sont modélisés par des produits matriciels.



Applications de l'algèbre linéaire

Prédiction linéaire

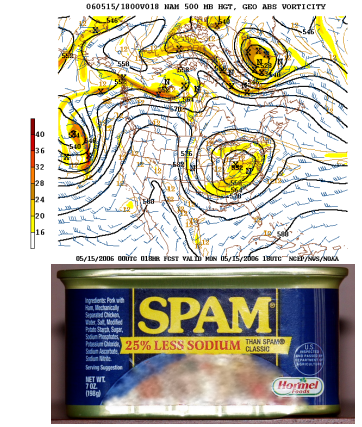
- Soit la fonction linéaire

$$f(\mathbf{x}) = \mathbf{v}^\top \mathbf{x} = \sum_k v_k x_k$$

- $\mathbf{x}$ est un vecteur décrivant un objet et $\mathbf{v}$ le paramètre du modèle.
- Prédiction de valeur continue.

Exemples d'utilisation

- Météo (température, humidité, direction du vent à partir de mesures).
- Moteur de recherche (score à partir de requête web et pagerank).
- Détection de SPAM (score à partir d'email).



Applications de l'algèbre linéaire

Factorisation de matrice

- On cherche à factoriser une matrice $\mathbf{A} \in \mathbb{R}^{m \times n}$ sous la forme :

$$\mathbf{A} \approx \mathbf{U}\mathbf{V}^\top$$

avec $\mathbf{U} \in \mathbb{R}^{m \times p}$, $\mathbf{V} \in \mathbb{R}^{n \times p}$ et $p \ll m, p \ll n$

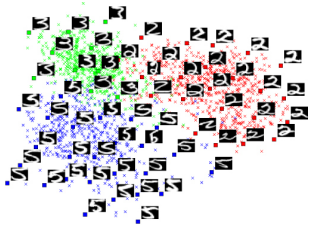
- $\mathbf{A}$ peut ne pas être complètement observée.

Exemples d'utilisation

- Recommandations Amazon.
- Recommandations Netflix.
- Clustering de réseaux sociaux.
- Réduction de dimension.

amazon.com

NETFLIX



Représentation mémoire en langage C

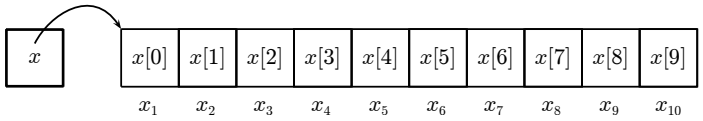
Représentation mémoire

- Il existe de multiples implémentations possibles (pointeurs, tableaux, ...).
- Dans ce cours nous nous concentrerons sur l'utilisation de pointeurs.
- Pour les matrices on veut un stockage mémoire contigu pour permettre une interface avec BLAS (interface standard pour les opérations d'algèbre linéaire).

Principe général

- Déclaration des vecteurs et matrices (pointeurs).
- Allocation mémoire dynamique au début du programme (`malloc`).
- Calculs effectués sur les vecteurs matrices alloués.
- Libération mémoire à la fin du programme (`free`).

Vecteurs en C



Principe

- Les vecteurs sont représentés par des pointeurs vers des flottants.
- Déclaration : `double *x;`
- Initialisation mémoire : `x=malloc(n*sizeof(double));` (`calloc` pour init. à 0)
- Utilisation : `x[i]=10.;` ou `*(x+i)=10.;`
- Libération mémoire : `free(x);`
- Attention à libérer la mémoire et à ne pas faire de `malloc` dans la boucle d'une procédure itérative.

Tableaux numpy en mémoire

- Numpy est une bibliothèque Python utilisant du C compilé.
- La majorité des opérations numériques sont accélérées en C.
- L'objet `np.array` est implémenté à travers une structure C.
- Le mème object peut gérer des vecteurs des matrice ou des tenseurs (d¿2).

Implémentation C

```c
typedef struct PyArrayObject {
  PyObject_HEAD
  char *data;
  int nd;
  npy_intp *dimensions;
  npy_intp *strides;
  PyObject *base;
  PyArray_Descr *descr;
  int flags;
  PyObject *weakreflist;
} PyArrayObject;
```

- `data` : pointeur vers la première valeur du tableau dans l'espace mémoire.
- `nd` : nombre de dimensions (=1 pour un vecteur).
- `dimensions` : pointeur vers les tailles de chaque dimension (une seule valeur pour un vecteur).
- `strides` : pas en mémoire pour parcourir le tableau

Tableau numpy : `np.array`

- Type objet de base pour tous les tableau numpy.
- Initialisé avec `np.array` ou les fonctions de création de tableaux `np.zeros,np.ones,...`.
- Implémenté avec tous les opérateurs mathématiques des base (`+,-,*,/,**`).

Attributs principaux

ndim Nb de dimensions du tableau (1D,2D,nD).

shape Taille du tableau dans chacune des dimensions.

size Nombre de valeurs au total.

dtype Type des valeurs dans le tableau.

nbytes Taille totale en octet du tableau.

data Pointeur vers la première valeur en mémoire.

Méthodes

sum() Retourne la somme de tous les éléments du tableau.

max() Valeur maximum du tableau (aussi `min()`).

argmax() Indexes des valeurs maximales.

reshape() Changer la taille du tableau.

copy() Copier le tableau dan n nouvel `np.array`.

Exercice 2 : Initialisation et impression

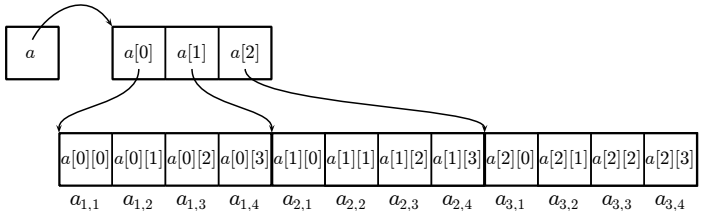
Quelle est la sortie imprimée dans le terminal pour le programme suivant :

Code source

```python
import numpy as np
n=5
v0=np.zeros(n)
v1=np.ones(n)
vf=np.full(n,3.14)
vr=np.arange(n)
vrand=np.random.randn(n)
v00=np.zeros_like(vr)
vl=np.array([1,2,3,6.0])
print("v0=",v0)
print("v1=",v1)
print("vf=",vf)
print("vr=",vr)
print("vrand=",vrand)
print("v00=",v00)
print("vl=",vl)
vr=2*vr+1
print("vr=",vr)
```

Sortie

Matrices en C



Principe

- Les matrices sont représentées par un tableau de pointeurs (vers chaque ligne de la matrice).
- Déclaration : `double *a;`
- Initialisation mémoire : `malloc` (taille du nombre d'élément dans la matrice)
- `stride` est le pas à fournir pour accéder à la ligne suivante en mémoire.
- Utilisation : `a[j+stride*i]=10.;`
- Libération mémoire (`free`) : `free(a);`
- Une espace mémoire contigu est nécessaire pour une interface avec BLAS.
- En Fortran (langage d'origine d BLAS) les matrice sont rangées en colonnes lieu de ligne.

Initialisation de matrices en numpy

- Les matrices en numpy sont des `np.array` 2D.
- Attention à ne pas utiliser le type `matrix` qui va être supprimé dans le futur.
- Numpy s'occupe de comment accéder aux valeur à partir de leur indice (opérateur `[]`).

Code source

```
1  import numpy as np
2  m,n=2,3
3  M0=np.zeros((m,n))
4  M1=np.ones((m,n))
5  Mr=np.arange(m*n).reshape((m,n))
6  Mrand=np.random.randn(m,n)
7  M00=np.zeros_like(Mr)
8  Ml=np.array([[1,2],[3,6.0]])
9  print("M0=",M0)
10 print("M1=",M1)
11 print("Mr=",Mr)
12 print("Mrand=",Mrand)
13 print("M00=",M00)
14 print("Ml=",Ml)
```

Sortie

```
1  $python3 ex_matrix.py
2  M0= [[0. 0. 0.]
3   [0. 0. 0.]]
4  M1= [[1. 1. 1.]
5   [1. 1. 1.]]
6  Mr= [[0 1 2]
7   [3 4 5]]
8  Mrand= [[-0.68628649 -0.79809459
         -0.03669245]
9   [-0.63345605 -1.15168145
         -1.21447778]]
10 M00= [[0 0 0]
11  [0 0 0]]
12 Ml= [[1. 2.]
13  [3. 6.]]
```


Matrices creuses

Principe

- Stockage mémoire classique est ≪ dense ≫.
- Les matrices et vecteurs creux (parcimonieux) ont un petit nombre $n_v \ll n$ de valeurs différentes de 0.
- Il est plus efficace de stocker en mémoire une liste des positions et les valeurs correspondantes.
- Vecteur $\mathbf{v}$ de taille n avec n_v valeurs différentes de zéros format compressé :

$$\mathbf{i}_s \in \mathbb{N}^{n_v} \text{ et } \mathbf{v}_s \in \mathbb{R}^{n_v}$$

- Matrice $\mathbf{A}$ de taille $m \times n$ avec n_v valeurs différentes de zéros format compressé :

$$\mathbf{i}_s \in \mathbb{N}^{n_v}, \quad \mathbf{j}_s \in \mathbb{N}^{n_v} \text{ et } \mathbf{v}_s \in \mathbb{R}^{n_v}$$

- La représentation ci-dessus s'appelle la représentation par coordonnée.
- Différents types de représentation (Yale sparse matrix, Compressed Sparse Column) sont plus ou moins efficaces selon les opérations à effectuer (opération matricielle, remplissage).

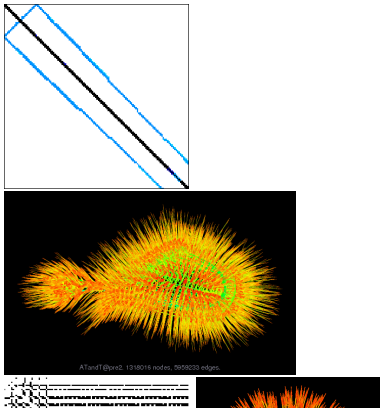
Matrice creuse (2)

Exemple sous Matlab

- $n = 10000$
- $n_v = 100$
- Temps de calcul (en sec).

Opération	Dense	Creux
$\mathbf{v}^\top \mathbf{v}$	0.0001	0.0005
$\mathbf{A}\mathbf{v}$	0.0525	0.0007
$\mathbf{A}\mathbf{A}$	19.2145	0.0397

Exemples d'utilisation



- Données web (page web, profil).
- Graphe de relations (réseaux sociaux).
- Objets 3D en infographie.

Matrices creuses en Python

- Les matrices creuses ne sont pas implémentées en numpy mais dans le module `scipy.sparse` de la bibliothèque scipy.
- Possibilité d'avoir de nombres format "sparses" et compatibilité avec les matrices denses de numpy.

Code source

```
1 import numpy as np
2 import scipy as sp
3 import scipy.sparse
4 m,n=5,5
5 M=sp.sparse.coo_matrix
      (([1,1.5,2],[(1,3,4),(2,1,0)]),
      shape=(m,n))
6 print(M)
7 print(M.todense())
8 print(M*2)
9 print((M*2).todense())
```

Sortie

```
1  $python3 ex_sparse.py
2    (1, 2) 1.0
3    (3, 1) 1.5
4    (4, 0) 2.0
5  [[0. 0. 0. 0. 0. ]
6   [0. 0. 1. 0. 0. ]
7   [0. 0. 0. 0. 0. ]
8   [0. 1.5 0. 0. 0. ]
9   [2. 0. 0. 0. 0. ]]
10   (1, 2) 2.0
11   (3, 1) 3.0
12   (4, 0) 4.0
13 [[0. 0. 0. 0. 0.]
14  [0. 0. 2. 0. 0.]
15  [0. 0. 0. 0. 0.]
16  [0. 3. 0. 0. 0.]
17  [4. 0. 0. 0. 0.]]
```

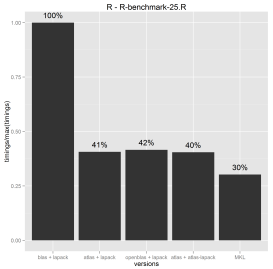

BLAS (Basic Linear Algebra Subprograms)

Description

- BLAS est une spécification décrivant les opérations de base en algèbre linéaire.
- L'origine de la spécification est une bibliothèque Fortran proposée en 1979.
- Une implémentation de référence en Fortran est disponible sur `http://www.netlib.org/blas/`.
- Types de fonctions séparées par leur complexité (mémoire/temporelle).
- Numpy utilise BLAS pour les opérations de base.
- Scipy fourni une interface blas dans le module `scipy.linalg.blas`.

Importance de la compatibilité

- De nombreuses bibliothèques fournissent une implémentation BLAS extrêmement optimisée.
- Choix de la bibliothèque au niveau système sous Linux.
- Exemples : Intel MKL, Atlas, OpenBLAS.



Source `http://blog.nguyenvq.com/`

BLAS et complexité

Les fonctions définies dans BLAS sont regroupées par niveaux de complexité algorithmique.

Niveau 1 (Complexité linéaire)

- Vecteur taille `n`.
- $T(n) = O(n)$, $S(n) = O(n)$
- Exemple : produit scalaire , multiplication par un scalaire, addition de vecteurs.

Niveau 2 (Complexité quadratique)

- Vecteur taille `n`, Matrice de taille `m`×`n`
- $T(n) = O(n^2)$ ou $O(mn)$, $S(n) = O(n^2)$ ou $O(mn)$
- Exemple : produit dyadique , multiplication matrice/vecteur.

Niveau 3 (Complexité cubique)

- Matrice de taille `n`×`n`
- $T(n) = O(n^3)$, $S(n) = O(n^2)$
- Exemple : produit matriciel.

Convention des fonctions BLAS

- Les fonctions et procédures BLAS sont des fonctions Fortran en majuscule.
- Nom typique : `xNOMFONCTION` où x donne le type des vecteurs :
  - **S** Simple précision (`float`)
  - **D** Double précision (`double`)
  - **C** Complexe simple précision
  - **Z** Complexe double précision
- L'interface `cblas` fournit des fonctions C appelant directement les fonctions BLAS.
- Nom des fonctions dans `scipy.linalg.blas` : `xnomfonction`
- Le passage de paramètre se fait toujours par référence pour les matrices et vecteurs (on donne les pointeurs)
- Les pointeurs doivent adresser des espaces mémoire alloués.
- Dans le cours nous donnerons le nom des fonctions BLAS associées aux opérations de base en algèbre linéaire.

Opérations de base sur des vecteurs

- Multiplication par un scalaire (xSCAL)
$$\mathbf{v} \leftarrow \alpha\mathbf{v}$$
- Somme et soustraction
$$\mathbf{v} \leftarrow \mathbf{x} + \mathbf{y}$$
- Produit scalaire (xDOT)
$$s \leftarrow \mathbf{x}^\top \mathbf{y}$$
- Cas général (xAXPY)
$$\mathbf{y} \leftarrow \alpha\mathbf{x} + \mathbf{y}$$
- Complexité de toutes ces opérations :
$$T(n) = O(n), \quad S(n) = O(n)$$

Numpy permet d'effectuer toutes ces opérations très simplement mais nous allons voir leur implémentation algorithmiques. Dans la suite oon supposera que blas a été import avec la commande suivante : `import scipy.linalg.blas as blas`

Opérations sur des vecteurs

Multiplication par un scalaire (xSCAL)

- $\mathbf{v} \leftarrow \alpha\mathbf{v}$
- Code numpy :
```
v=alpha*v
v[:]=alpha*v
```
- Fonction BLAS de Niveau 1 :
`blas.dscal(alpha,v)`

Code
```
def dscal(alpha,v):
    """v=alpha*v"""
    for i,vi in enumerate(v):
        v[i]=alpha*vi
    return v
```

Attention ! Modification du vecteur `v` à l'intérieur de la fonction.

Produit scalaire (xDOT)

- $s \leftarrow \mathbf{x}^\top \mathbf{y}$
- Code numpy :
```
s=x.dot(y)
s=np.dot(x,y)
```
- Fonction BLAS de Niveau 1 :
`s=blas.ddot(x,y)`

Code
```
def ddot(x,y):
    """s=x^Tv"""
    res=0
    for xi,yi in zip(x,y):
        res+=xi*yi
    return res
```


Exercice 3 : xAXPY

Code
```
def daxpy(x,y,a=1):
    "y=a*x+y"



    return y
```

Exercice

- Compléter la fonction xAXPY à gauche.
- On supose `n` déclaré et les vecteurs `x` et `y` initialisés.
- Montrer que cette fonction permet d'effectuer des sommes, soustractions et multiplication par un scalaire :
  - Somme $\mathbf{y} \leftarrow \mathbf{x} + \mathbf{y}$:
  - Soustraction $\mathbf{y} \leftarrow \mathbf{y} - \mathbf{x}$:
  - Mult. scalaire $\mathbf{y} \leftarrow \alpha\mathbf{x}$:

- Code numpy :
```
y=a*x+y
```
- Fonction BLAS de Niveau 1 :
`blas.daxpy(x,y,a=a)`

Opérations matrice/vecteur (niveau 2)

Mise à jour de rang 1 (xGER)

$$\mathbf{A} \leftarrow \alpha\mathbf{x}\mathbf{y}^\top + \mathbf{A}$$

- Avec $\alpha \in \mathbb{R}$, $\mathbf{y} \in \mathbb{R}^n$, $\mathbf{x} \in \mathbb{R}^m$ et $\mathbf{A} \in \mathbb{R}^{m\times n}$.
- Complexité : $T(n) = O(mn), \quad S(n) = O(mn)$
- Cas particulier : $\mathbf{A}$ symétrique et $\mathbf{x} = \mathbf{y}$ (xSYR).

Produit matrice vecteur généralisé (xGEMV)

$$\mathbf{y} \leftarrow \alpha\mathbf{A}\mathbf{x} + \beta\mathbf{y}$$

- Avec $\alpha, \beta \in \mathbb{R}^2$, $\mathbf{y} \in \mathbb{R}^m$, $\mathbf{x} \in \mathbb{R}^n$ et $\mathbf{A} \in \mathbb{R}^{m\times n}$.
- Complexité :
- Cas particuliers :
  - $\mathbf{A}$ symétrique (xSYMV).
  - $\mathbf{A}$ bande (xSBMV).
  - $\mathbf{A}$ triangulaire (xSTRMV).

Mise à jour de rang 1 (xGER)

Implémentation

$$\mathbf{A} \leftarrow \alpha \mathbf{x}\mathbf{y}^\top + \mathbf{A}$$

- $\alpha \in \mathbb{R}$, $\mathbf{y} \in \mathbb{R}^n$, $\mathbf{x} \in \mathbb{R}^m$ et $\mathbf{A} \in \mathbb{R}^{m \times n}$.
- Code numpy :

```
A+=alpha*np.outer(x,y)
A=A+alpha*np.outer(x,y)
A=A+alpha*x[:,None]*y[None,:]
```

- Fonction BLAS de Niveau 2 :

```
A=dger(alpha,x,y,a=A)
dger(alpha,x,y,a=A,overwrite_a=True)
```

Code

```
def dger(alpha,x,y,a=None):
    "a=a+alpha*x*y^T"
    if a is None:
        a=np.zeros((len(x),len(y)))
    for i,xi in enumerate(x):
        for j,yj in enumerate(y):
            a[i,j]+=alpha*xi*yj
    return a
```


Exercice 4 : Produit généralisé (xGEMV)

Exercice

$$\mathbf{y} \leftarrow \alpha \mathbf{A}\mathbf{x} + \beta \mathbf{y}$$

- $\alpha, \beta \in \mathbb{R}^2$, $\mathbf{y} \in \mathbb{R}^m$, $\mathbf{x} \in \mathbb{R}^n$ et $\mathbf{A} \in \mathbb{R}^{m \times n}$.
- Compléter la fonction xGEMV à gauche.
- Code numpy :

```
y=alpha*A.dot(x)+beta*y
y[:]=alpha*A.dot(x)+beta*y
```

- Fonction BLAS de Niveau 2 :

```
y=blas.dgemv(alpha,A,x,beta,y)
blas.dgemv(alpha,A,x,beta,y,
overwrite_y=true)
```


Opération matricielle généralisée (xGEMM)

Opération

$$\mathbf{C} \leftarrow \alpha \mathbf{A}\mathbf{B} + \beta \mathbf{C}$$

- $\alpha, \beta \in \mathbb{R}^2$, $\mathbf{C} \in \mathbb{R}^{m \times n}$, $\mathbf{A} \in \mathbb{R}^{m \times p}$, $\mathbf{B} \in \mathbb{R}^{p \times n}$
- Complexité : $T(n) = O(mnp), \quad S(n) = O(mn)$

Cas particuliers

- $\mathbf{A}$ symétrique (xSYMM)
- $\mathbf{A}$ hermitienne (xHEMM)
- $\mathbf{A}$ triangulaire (xTRMM)
- $\mathbf{B} = \mathbf{A}^\top$ (xSYRK)
- $\mathbf{C} \leftarrow \alpha \mathbf{A}\mathbf{B}^\top + \bar{\alpha} \mathbf{B}\mathbf{A}^\top + \beta \mathbf{C}$ (mise à jour symétrique xSYRK2)

Opération matricielle généralisée (xGEMM)

Implémentation

$$\mathbf{C} \leftarrow \alpha \mathbf{A}\mathbf{B} + \beta \mathbf{C}$$

- $\alpha, \beta \in \mathbb{R}^2$, $\mathbf{C} \in \mathbb{R}^{m \times n}$, $\mathbf{A} \in \mathbb{R}^{m \times p}$, $\mathbf{B} \in \mathbb{R}^{p \times n}$
- Code numpy :

```
y=alpha*A.dot(x)+beta*y
y[:]=alpha*A.dot(x)+beta*y
```

- Fonction BLAS de Niveau 3 :

```
C=blas.dgemm(alpha,A,B,beta,C)
blas.dgemm(alpha,A,B,beta,C,
overwrite_c=True)
```

Code

```
def dgemm(alpha,A,B,beta=0,C=None):
    """C=alpha*A*B+beta*C"""
    if C is None:
        C=np.zeros((A.shape[0],B.shape[1]))
    for i in range(A.shape[0]):
        for j in range(B.shape[1]):
            C[i,j]*=beta
            for k in range(A.shape[1]):
                C[i,j]+=A[i,k]*B[k,j]
    return C
# ou
def dgemm(alpha,A,B,beta=0,C=None):
    """C=alpha*A*B+beta*C"""
    if C is None:
        C=np.zeros((A.shape[0],B.shape[1]))
    for i in range(A.shape[0]):
        for j in range(B.shape[1]):
            C[i,j]*=beta
            C[i,j]+=A[i,:].dot(B[:,j])
    return C
```

Implémentation efficace en Numpy

Vectorisation d'opérations

- Python est interprété, les boucles sont à éviter avec les tableaux numpy.
- Préférer les formulations "vectorisées" : `a+2*b,np.exp(a)`
- Les fonctions numpy peuvent être parallélisées à travers BLAS : `np.dot`
- Penser à coder des fonctions génériques (qui calculent le résultat en vectorisé).

Attention mémoire et pointeur

- Numpy minimise les allocations mémoire : `b=a,b[0]=1` ne pas copie en mémoire et modifie `b` et `a`.
- Le "slicing" n'effectue pas non plus de copies : `b=A[:,1],b[0]=1`
- Utiliser la copie explicite pour être sûr : `b=a.copy()`.

Cython

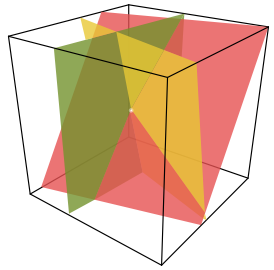
- Si les boucles sont nécessaire le langage Cython est un Python typé qui permet de compiler directement en c.
- Syntaxe très similaire, peut utiliser OpenMP pour faire des calculs parallèle.

Système d'équations linéaires

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 + ... + a_{1,n}x_n = b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + ... + a_{2,n}x_n = b_2 \\ \vdots \\ \vdots \\ a_{m,1}x_1 + a_{m,2}x_2 + ... + a_{m,n}x_n = b_m \end{cases} \Leftrightarrow \quad \mathbf{Ax} = \mathbf{b}$$

avec $\mathbf{A} \in \mathbb{R}^{m \times n}$ et $\mathbf{b} \in \mathbb{R}^m$.

- On cherche le vecteur $\mathbf{x}$ de taille n satisfaisant les m équations ci dessus.
- L'ensemble des solutions dépend des propriétés de la matrice $\mathbf{A}$.
- La résolution du système peut être particulièrement simple dans certains cas particuliers ($\mathbf{A}$ diagonale, triangulaire).



Solutions des systèmes d'équations linéaires

Existence et unicité des solutions

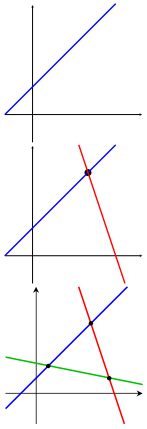
Dépend de m, n et du rang de $\mathbf{A}$:

- $m < n$, il existe une infinité de solution.
- $\mathbf{A}$ de rang $< n$ il existe une infinité de solution.
- $m = n$ et $\mathbf{A}$ de rang n, il existe une solution unique
- $\mathbf{A}$ de rang $> n$, il n'existe pas de solution
  $\rightarrow$ solution des moindres carrés.

Résolution du système

- Cas particulier où $m = n$ et $\mathbf{A}$ de rang n
- Solution unique :
$$\mathbf{x}^\star = \mathbf{A}^{-1}\mathbf{b}$$
- Éviter l'inversion matricielle directe dès que possible (matrices particulières).

Exemple $n = 3$



LAPACK (Linear Algebra PACKage)

Description

- LAPACK est une bibliothèque Fortran qui fournit des fonctions de haut niveau en algèbre linéaire.
- Bibliothèque d'accompagnement de BLAS utilisant les routines BLAS le plus possible.
- Interface Python directe dans le module `scipy.linalg.lapack`.
- Fonctions disponibles dans `numpy.linalg` et `scipy.linalg`

L	A	P	A	C	K
L	-A	P	-A	C	-K
L	A	P	A	-C	-K
L	-A	P	-A	-C	K
L	A	-P	-A	C	K
L	-A	-P	A	C	-K

Fonctions proposées (Niveau 3)

- Résolution de systèmes linéaires (cas général).
- Estimation des moindres carrés.
- Factorisation de matrice (EIG, SVD, LU, QR).

LAPACK et interface Python

Convention de nommage

- Nom des fonctions LAPACK : `XMMAAA`

  X type de données (même que pour BLAS).
  MM type de matrice (`GE` générale, `GB` bande, `SY` symétrique, `TR` triangulaire, `DI` diagonale ...
  AAA opération de taille 2-3 caractères.

- Type d'opérations `AAA` supportées :
  - Résolution de systèmes linéaires (`SV`).
  - Moindres carrés (`LS`).
  - Décomposition en valeurs/vecteurs propres EIG (`TRD`).
  - Décomposition LU (`TRF`).
  - Décomposition en valeur singulière SVD (`BRD`).

Lapack dans `scipy`

- Nom des fonctions : `scipy.linalg.lapack.xmmaaa`.

Matrice $\mathbf{A}$ diagonale

Résolution du système

- n équations indépendantes de la forme
$$a_{k,k}x_k = b_k$$
- Solutions évidentes de la forme
$$x_k = \frac{b_k}{a_{k,k}}$$

$$\begin{bmatrix} a_{1,1} & 0 & \cdots & 0 \\ 0 & a_{2,2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{n,n} \end{bmatrix}$$

- Si $a_{i,i} = 0$ et $b_i \neq 0$ le système n'admet pas de solution.
- Si $a_{i,i} = 0$ et $b_i = 0$ le système admet une infinité de solutions.

Code

```
1 def diag_sv(A,b):
2     """x=A\b, A diagonal"""
3     x=np.zeros(A.shape[0])
4     for k in range(A.shape[0]):
5         x[k]=b[k]/A[k,k]
6     return x
```

Discussion

- Complexité $T(n)$:
- Complexité $S(n)$:

Matrice triangulaire

Résolution du système

- Soit $\mathbf{A} = \mathbf{L}$ une matrice triangulaire inférieure.
- Le système linéaire peut être résolu composante par composante :

$$x_1 = \frac{b_1}{l_{1,1}}$$
$$x_2 = \frac{b_2 - l_{2,1}x_1}{l_{2,2}}$$
$$x_k = \frac{b_k - \sum_{i=1}^{k-1} l_{k,i}x_i}{l_{k,k}}$$

Matrice

$$\mathbf{L} = \begin{bmatrix} l_{1,1} & 0 & \cdots & 0 \\ l_{2,1} & l_{2,2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n,1} & l_{n,2} & \cdots & l_{n,n} \end{bmatrix}$$

Système

$$\begin{aligned} l_{1,1}x_1 &= b_1 \\ l_{2,1}x_1 + l_{2,2}x_2 &= b_2 \\ &\vdots \\ l_{k,1}x_1 + \cdots + l_{k,k}x_k &= b_k \end{aligned}$$

Exercice 5 :

Trouver la solution $\mathbf{x}$ pour le système suivant

$$\mathbf{L} = \begin{bmatrix} 2 & 0 & 0 \\ 4 & -1 & 0 \\ -6 & 1 & -2 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 2 \\ 3 \\ -7 \end{bmatrix}$$

Solution

$$\mathbf{x} =$$

Exercice 6 : Résolution de système triangulaire

Code

```
1  def tri_inf_sv(L,b):
2      """x=L\b, L tri inf"""
3
4
5
6
7
8
9
10
11     return x
```

Exercice

$$\mathbf{x} \leftarrow \mathbf{L}^{-1}\mathbf{b}$$

avec $\mathbf{L}$ triangulaire inférieure.

- Compléter la fonction `triinfsv` à gauche.
- Complexité $T(n)$: $O(n^2)$
- Complexité $S(n)$: $O(n^2)$
- Serait-il possible de coder une résolution plus efficace en mémoire qui utilise $\mathbf{b}$ pour écrire la solution ?

Matrice A générale : Pivot de Gauss

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 + ... + a_{1,n}x_n = b_1 & (L_1) \\ a_{2,1}x_1 + a_{2,2}x_2 + ... + a_{2,n}x_n = b_2 & (L_2) \\ \vdots & \vdots \\ a_{m,1}x_1 + a_{m,2}x_2 + ... + a_{m,n}x_n = b_m & (L_m) \end{cases}$$

- Effectuer des combinaisons linéaires des équations ne change pas le problème.
- Le pivot de Gauss vise à utiliser des sommes pondérées des équations pour faire disparaître les variables une à une.
- A la fin, on trouve un système linéaire triangulaire simple à résoudre ($O(n^2)$) .
- Première étape (colonne 1) : $L_2 \leftarrow L_2 - \frac{a_{2,1}}{a_{1,1}}L_1$, ..., $L_m \leftarrow L_m - \frac{a_{m,1}}{a_{1,1}}L_1$

$$\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 + ... + a_{1,n}x_n &= b_1 \\ 0 + (a_{2,2} - \frac{a_{2,1}a_{1,2}}{a_{1,1}})x_2 + ... + (a_{2,n} - \frac{a_{2,1}a_{1,n}}{a_{1,1}})x_n &= b_2 - \frac{a_{2,1}}{a_{1,1}}b_1 \\ 0 + (a_{m,2} - \frac{a_{m,1}a_{1,2}}{a_{1,1}})x_2 + ... + (a_{m,n} - \frac{a_{m,1}a_{1,n}}{a_{1,1}})x_n &= b_m - \frac{a_{m,1}}{a_{1,1}}b_1 \end{aligned}$$

Exemple pivot de Gauss

Résolution

- $n = 3$.

Système linéaire

$$\begin{aligned} 2x + y - z &= 8 && (L_1) \\ -3x - y + 2z &= -11 && (L_2) \\ -2x + y + 2z &= -3 && (L_3) \end{aligned}$$

Matrice augmentée

$$\left[\begin{array}{ccc|c} 2 & 1 & -1 & 8 \\ -3 & -1 & 2 & -11 \\ -2 & 1 & 2 & -3 \end{array}\right]$$

Itération 1

$L_2 \leftarrow L_2 + \frac{3}{2}L_1$

$L_3 \leftarrow L_3 + L_1$

$$\begin{aligned} 2x + y - z &= 8 && (L_1) \\ \frac{1}{2}y + \frac{1}{2}z &= 1 && (L_2) \\ 2y + z &= 5 && (L_3) \end{aligned}$$

$$\left[\begin{array}{ccc|c} 2 & 1 & -1 & 8 \\ 0 & 1/2 & 1/2 & 1 \\ 0 & 2 & 1 & 5 \end{array}\right]$$

Itération 2

$L_3 \leftarrow L_3 - 4L_2$

$$\begin{aligned} 2x + y - z &= 8 && (L_1) \\ \frac{1}{2}y + \frac{1}{2}z &= 1 && (L_2) \\ -z &= 1 && (L_3) \end{aligned}$$

$$\left[\begin{array}{ccc|c} 2 & 1 & -1 & 8 \\ 0 & 1/2 & 1/2 & 1 \\ 0 & 0 & -1 & 1 \end{array}\right]$$

La matrice est maintenant sous forme triangulaire.

Pivot de Gauss en Python

Implémentation simplifiée

$$\mathbf{A}, \mathbf{b} \leftarrow \text{pivot}(\mathbf{A}, \mathbf{b})$$

- $\mathbf{A} \in \mathbb{R}^{n \times n}$, $\mathbf{b} \in \mathbb{R}^n$
- Complexité $T(n)$:
- Pour résoudre le système linéaire :
  1. $\mathbf{A}, \mathbf{b} \leftarrow \text{pivot}(\mathbf{A}, \mathbf{b})$
  2. $\mathbf{b} \leftarrow \mathbf{A}^{-1}\mathbf{b}$: inversion de matrice triangulaire.
- Résolution en Python :

```
pivot(A,b)
x=tri_sup_sv(A,b)
```

Code

```
def pivot(A,b):
    """Pivot de Gauss"""
    n=A.shape[0]
    for k in range(n-1):
        # colone k a mettre en echelon
        for i in range(k+1,n):
            for j in range(k+1,n):
                A[i,j]-=A[k,j]*A[i,k]/A[k,k]
            # mise a jour b
            b[i]-=A[i,k]*b[k]/A[k,k]
            A[i,k]=0 # pivot a 0
    return A,b
```


Erreur numérique et implémentation

Erreur numérique et permutation

- La méthode précédente traite les données lignes par lignes.
- Attention à la propagation de l'erreur pour des flottants !
- Erreur amplifiée lorsque le pivot est petit (division par le pivot).
- En pratique on permute les lignes le long des itérations pour prendre le pivot ayant la valeur absolue maximum (Lapack retourne la permutation).

Implémentation Python

- Lapack implémente la résolution de système générale à l'aide d'une factorisation LU (variante du pivot de Gauss).
- Code numpy :

```
x=np.linalg.solve(A,b)
```

- Fonction Lapack xDGESV (niveau 3)

```
lu,piv,x,info=lapack.dgesv(A,b)
x=lapack.dgesv(A,b)[2]
```

Applications du pivot de Gauss

Calcul de déterminant

- Le déterminant de $\mathbf{A}$ est égale au déterminant de la matrice triangulaire (multiplication de ses éléments diagonaux).
- Règles pour le calcul du déterminant :
  - Multiplier une ligne par un scalaire multiplie le déterminant par un scalaire.
  - Permuter deux lignes multiplie le déterminant par -1.
  - Ajouter à une ligne une combinaison linéaire des autres ne change pas le déterminant.

Calcul de l'inverse d'une matrice

- On utilise une matrice augmentée $[\mathbf{A}|\mathbf{I}]$ avec la matrice identité $\mathbf{I}$.
- Après triangualisation, on utilise une backsubstitution pour retrouver une matrice identité à gauche. La matrice augmentée devient $[\mathbf{I}|\mathbf{A}^{-1}]$.

Calcul du rang d'une matrice

- Si une ligne nulle apparaît le long des itérations, la matrice n'est pas inversible ($\text{rank}(\mathbf{A}) < n$).
- Le rang de la matrice est le nombre de lignes non-nulles.

Linpack et HPLinpack

- La résolution de système linéaire par pivot de Gauss a une complexité $O(n^3)$.
- Algorithme utilisé dans les systèmes de $\ll$ benchmark $\gg$ Linpack et HPLinpack pour estimer la puissance de calcul des gros centres de calculs dans le monde.
- Un classement des centres de calculs mondiaux est disponible sur le site : `http://www.top500.org/`

Ordres de grandeur (Juin 2015)

Machine	Flops
PC perso, Intel Core i7	3.10^9
Station de travail, Intel Xeon 12 coeurs	10.10^9
Station de travail, NVIDIA Titan X (Cuda)	2.10^{12}
Centre de calcul de l'université de Nice	35.10^{12}
Total Exploration Production, France (rang 36)	2.10^{15}
National Super Computer Center in Guangzhou, China (rang 1)	33.10^{15}
Cerveau humain (simulation temps réel)	37.10^{15}

Rappels : 10^9 (Giga), 10^{12} (Tera), 10^{15} (Peta)

Factorisation de matrice

Principe

- On cherche à représenter $\mathbf{A}$ comme une multiplication de plusieurs matrices plus simples (triangulaires).
- Permet de résoudre de multiples systèmes linéaires plus efficacement.
- Meilleur interprétation des propriétés de $\mathbf{A}$.

Factorisations classiques

LU $\mathbf{A} = \mathbf{LU}$, où $\mathbf{L}$ triangulaire inférieure et $\mathbf{U}$ triangulaire supérieure.

Cholesky $\mathbf{A} = \mathbf{LL}^\top$, pour $\mathbf{A}$ symétrique définie positive.

QR $\mathbf{A} = \mathbf{QR}$, avec $\mathbf{Q}$ orthogonale et $\mathbf{R}$ triangulaire supérieure.

Vec. propres $\mathbf{A} = \mathbf{VDV}^\top$, où $\mathbf{V}$ orthonormale (vec. propres) et $\mathbf{D}$ diagonale (val. propres).

SVD $\mathbf{A} = \mathbf{UDV}^\top$, où $\mathbf{U}$ et $\mathbf{V}$ orthonormales et $\mathbf{D}$ diagonale.

Factorisation LU

Principe

$$\mathbf{A} = \mathbf{LU}, \qquad \text{avec } \mathbf{A} \in \mathbb{R}^{n\times n}, \mathbf{L} \in \mathbb{R}^{n\times n} \text{ tri. inf. et } \mathbf{U} \in \mathbb{R}^{n\times n} \text{ tri. sup.}$$

- Méthode proposée et étudiée par Alan Turing en 1948.
- Théorème d'existence : Si $\mathbf{A}$ est non singulière, il existe une décomposition $\mathbf{LU}$.
- Une généralisation permet d'utiliser une permutation des lignes : $\mathbf{PA} = \mathbf{LU}$
- On applique le pivot de Gauss sur ma matrice augmentée $[\mathbf{A}|\mathbf{I}]$ pour trouver une matrice triangulaire supérieure $[\mathbf{U}|\cdot]$.
- Complexité $T(n) = \frac{2n^3}{3}$.

Résolution de systèmes linéaires $\mathbf{Ax} = \mathbf{b}$

- $\mathbf{Ax} = \mathbf{L}(\mathbf{Ux}) = \mathbf{b} \rightarrow \mathbf{Ly} = \mathbf{b} \rightarrow \mathbf{Ux} = \mathbf{y}$
- $\mathbf{x} = \mathbf{U}^{-1}\mathbf{L}^{-1}\mathbf{b}$, soit deux systèmes linéaires triangulaires.

Implémentation Lapack : `DGETRF`

- Appel en Scipy : `P,L,U=scipy.linalg.lu(A)`
- Appel en Lapack : `LU,piv,info=lapack.dgetrf(A)` (LU : L et U superposée)
- Utilisé pour la résolution de systèmes linéaires généralisés avec `DGESV`.

Factorisation de Cholesky

Principe

$$\mathbf{A} = \mathbf{L}\mathbf{L}^\top, \qquad \text{avec } \mathbf{A} \in \mathbb{R}^{n\times n} \text{sym def. pos.}, \mathbf{L} \in \mathbb{R}^{n\times n} \text{ tri. inf.}$$

- Méthode proposée en 1924 par André-Louis Cholesky.
- Matrices carrées définies positives mais généralisée aux matrices complexes avec le transposé conjugué.
- Expression alt. : $\mathbf{A} = \mathbf{L}\mathbf{D}\mathbf{L}^\top$ avec $\mathbf{D}$ diagonale et $\mathbf{L}$ unitaire.
- Complexité $T(n) = \frac{n^3}{3}$ ($O(n^3)$ mais 2× plus efficace que LU).

Résolution de systèmes linéaires $\mathbf{A}\mathbf{x} = \mathbf{b}$

- $\mathbf{A}\mathbf{x} = \mathbf{L}(\mathbf{L}^\top\mathbf{x}) = \mathbf{b} \rightarrow \mathbf{L}\mathbf{y} = \mathbf{b} \rightarrow \mathbf{L}^\top\mathbf{x} = \mathbf{y}$
- $\mathbf{x} = \mathbf{L}^{\top-1}\mathbf{L}^{-1}\mathbf{b}$, soit deux systèmes linéaires triangulaires.

Implémentation Lapack : DPBTRF

- Appel en Scipy : `L = scipy.linalg.cholesky(a,lower=True)`
- Appel en Lapack : `LU,info=lapack.dpbtrf(A)` (LU : L et U superposée)
- Utilisé pour la résolution de systèmes linéaire généralisés avec DPPTRS.

Factorisation QR

Principe

$$\mathbf{A} = \mathbf{Q}\mathbf{R}^\top, \qquad \text{avec } \mathbf{Q} \in \mathbb{R}^{n\times n} \text{orthonormale}, \mathbf{R} \in \mathbb{R}^{n\times n} \text{ tri. sup.}$$

- Découverte indépendemment par John G. F. Francis en 1959, et Vera Kublanovskaya en 1961.
- Si $\mathbf{A}$ inversible alors factorisation unique.
- Factorisation d'une matrice rectangulaire avec $m > n$, alors les dernières lignes de $\mathbf{R}$ sont nulles.
- Permet d'estimer le déterminant de $\mathbf{A}$ à partir du déterminant de $\mathbf{R}$.
- Complexité (Householder) $T(n) = \frac{2}{3}n^3 + n^2 + \frac{1}{3}n - 2 = O(n^3)$.

Résolution de systèmes linéaires $\mathbf{A}\mathbf{x} = \mathbf{b}$

- $\mathbf{A}\mathbf{x} = \mathbf{Q}(\mathbf{R}\mathbf{x}) = \mathbf{b} \rightarrow \mathbf{y} = \mathbf{Q}^{-1}\mathbf{b} = \mathbf{Q}^\top\mathbf{b} \rightarrow \mathbf{R}\mathbf{x} = \mathbf{y}$
- $\mathbf{x} = \mathbf{R}^{-1}\mathbf{Q}^\top\mathbf{b}$, soit une multiplication et un système linéaire triangulaire.

Implémentation Lapack : DGEQRF

- Appel en Scipy : `Q,R = scipy.linalg.qr(A)`
- Appel en Lapack : `QR=lapack.dgeqrf(A)[0]` (QR : Q et R superposée)

Factorisation SVD

Principe de la décomposition en valeurs singulières

$\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^\top$ avec $\mathbf{A} \in \mathbb{R}^{m\times n}$ $\mathbf{U} \in \mathbb{R}^{m\times m}$ $\mathbf{V} \in \mathbb{R}^{n\times n}$ othonormales, $\mathbf{D} \in \mathbb{R}^{m\times n}$ diag.

- $\mathbf{A}$ peut être une matrice quelconque $m \neq n$.
- $\mathbf{D} = \text{diag}(\sigma_1, \ldots, \sigma_r)$ avec $r = min(m, n)$ contient les valeurs singulières.
- Calculée en effectuant plusieurs itérations de factorisation QR.
- Complexité $T(n) = km^2n + k'n^3 = O(n^3 + m^2n)$ avec $k = 4$ et $k' = 22$.

Approximation faible rang de la matrice $\mathbf{A}$

La meilleur approximation de rang $k < r$ de la matrice $\mathbf{A}$ obtenue par

$$\widetilde{\mathbf{A}} = \mathbf{U}\widetilde{\mathbf{D}}\mathbf{V}^\top \quad \text{où } \widetilde{\mathbf{D}} = \text{diag}(\sigma_1, \ldots, \sigma_k, 0, \ldots, 0) \quad (\text{ mise à 0 des } \sigma_i \text{ après } k)$$

Implémentation Lapack : DGESVD

- Appel en Scipy : `U,d,Vt = scipy.linalg.svd(A)`
- Appel en Lapack : `U,d,Vt=lapack.dgesvd(A)`

Ressources bibliographiques I

- Rappels algèbre linéaire [3].
- Référence complète sur les matrices [2].
- Implémentation en C [4].
- Doc BLAS/LAPACK [1].

[1] "Reference manual for intel® math kernel library - c," https://software.intel.com/en-us/mkl-reference-manual-for-c, version du 2015-11-16.

[2] G. H. Golub and C. F. Van Loan, *Matrix computations*. JHU Press, 2012, vol. 3.

[3] J. Grifone, *Algebre linéaire*. Cepadues-Editions, 1990.

[4] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical recipies in C*. Cambridge university press Cambridge, UK, 1992.