Introduction au calcul numérique

Camille Coti

LIPN, CNRS-UMR7030, Université Paris 13, F-93430 Villetaneuse, France

8 avril 2011

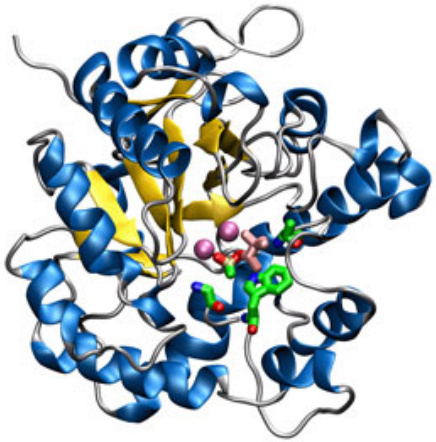
Roadmap

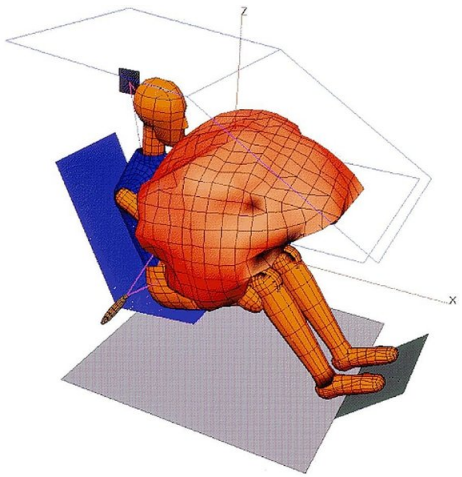
Introduction

Algèbre linéaire

L'algèbre linéaire a de nombreuses applications numériques

- Simulations : biologie *in-silico*, dynamique moléculaire, mécanique...
- Résolution d'équations
- Régression linéaire et moindres carrés...





Calcul numérique sur ordinateur

On veut utiliser la puissance des ordinateurs pour résoudre ces problèmes

- Comment représenter ces problèmes de façon **compréhensible par une machine** ?
- **Comment résoudre** ces problèmes ?
- Comment les résoudre **efficacement** ?

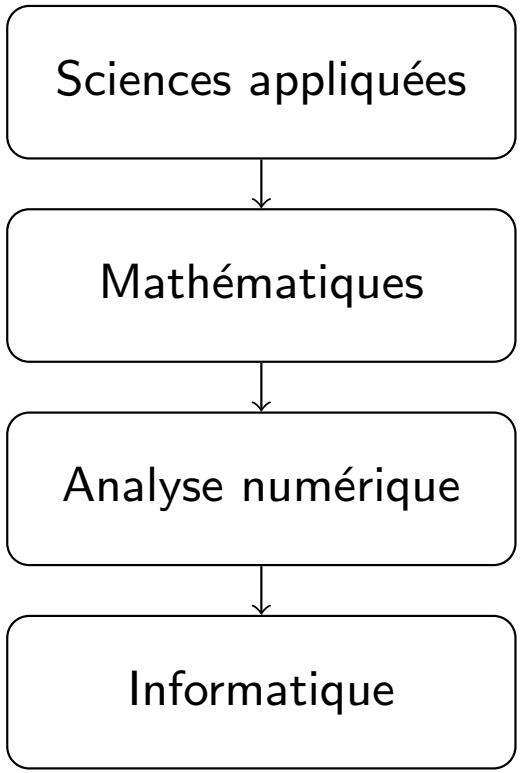
Démarche de résolution d'un problème

Problème de départ

Mise en équations (modélisation)

Discrétisation

Calcul



Opérations de base

Noyaux de calcul

On retrouve certaines opérations très fréquemment

- On cherche à les optimiser particulièrement
- Les applications appellent ces **noyaux** de calcul pour résoudre les problèmes

Exemples

Quelques exemples :

- **Transformée de Fourier** : traitement du signal, traitement d'images, analyse fréquentielle...
- **Multiplication matrice-vecteur** : probabilités (vecteur de variables stochastiques), théorie des files d'attentes...
- **Multiplication matrice-matrice** : 3D, application d'une fonction (*e.g.*, filtrage d'un signal), optique (matrice de réfraction / matrice de translation)...
- **Factorisations** : résolution de systèmes, décomposition en éléments propres, moindres carrés, obtention d'une base orthonormée...

Transformée de Fourier

Définition (Transformée de Fourier)

La **transformée de Fourier** d'une fonction f intégrable sur $\mathbb{R}$ est une opération telle que :
$F(f) : \nu \mapsto \hat{f}(\nu) = \int_{-\infty}^{\infty} f(\nu) e^{-2i\pi\nu t} \, \mathrm{d}t$

Un ordinateur peut calculer la transformée de Fourier **discrète** d'un vecteur.

Définition (Transformée de Fourier discrète)

La **transformée de Fourier discrète** d'un vecteur s de N échantillons est une opération telle que :
$S(k) = \sum_{n=0}^{N-1} s_n e^{-2i\pi k \frac{n}{N}}$ pour $0 \leq k < N$
En notant ζ_N^k les racines de l'unité : $\zeta_N^k = e^{-2i\pi \frac{k}{N}}$, on obtient :
$S(k) = \sum_{n=0}^{N-1} s_n \zeta_N^{kn}$ pour $0 \leq k < N$

Le calcul de la transformée de Fourier discrète d'un vecteur s consiste donc à calculer l'ensemble des coefficients S.

Nombre d'opérations de calcul

Définition (Complexité)

La **complexité** d'un algorithme est le nombre d'opérations que la machine doit effectuer pour l'exécuter.

Complexité du calcul de la transformée de Fourier

- On doit calculer N coefficients $S(k)$ (k allant de 0 à $N-1$)
- Chaque coefficient est calculé par une somme de N termes $S(k) = \sum_{n=0}^{N-1} s_n \zeta_N^{kn}$

Donc le calcul de la transformée de Fourier discrète nécessite $N \times N$ additions et $N \times N$ multiplications.

Théorème

La complexité du calcul de la transformée de Fourier discrète d'un signal vectoriel composé de N points est en $O(N^2)$.

Transformée de Fourier rapide

Une complexité en $O(N^2)$ est trop élevée :

- 100 opérations pour un vecteur de 10 éléments
- 10 000 opérations pour un vecteur de 100 éléments
- 1 000 000 opérations pour un vecteur de 1 000 éléments !


On cherche à réduire cette complexité


Utilisation de sommes partielles

Décomposition du calcul des indices

Principe : on divise le vecteur en deux parties entrelacées.
Par exemple : d'un côté les indices de rang pair, de l'autre les indices de rang impair. On a alors :
$S(k) = \sum_{n=0}^{N/2-1} s_{2n} e^{-2i\pi k \frac{2n}{N}} + \sum_{n=0}^{N/2-1} s_{2n+1} e^{-2i\pi k \frac{2n+1}{N}}$
Récursivement, on peut encore découper ces sommes en deux parties.

Algorithme de Cooley-Tukey

Algorithme de Cooley-Tukey

```
  fonction FFT( V, N ):    // vecteur V de longueur N
     si N = 1 alors:
        retourner V
     sinon:
        FFT( V[0, N/2-1], N/2 )
        FFT( V[N/2, N1], N/2 )
        pour k de 0 à N/2 - 1:
           tmp = V[k]
           V[k] = tmp + exp(-2 Pi i k/N) V[k+N/2]
           V[k+N/2] = tmp - exp(-2 Pi i k/N) V[k+N/2]
        fin pour
     fin si
fin
```

Théorème (Complexité de la transformée de Fourier rapide)

La complexité du calcul de la transformée de Fourier discrète d'un signal vectoriel composé de N points en utilisant l'algorithme de Cooley-Tukey est en $O(Nlog_2N)$.

Transformée de Fourier en 3D

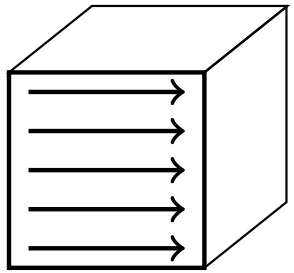
On peut calculer la transformée de Fourier en plusieurs dimensions d'un signal multi-dimensionnel.
Applications : électromagnétique, optique, cristallographie...

Fonctionnement

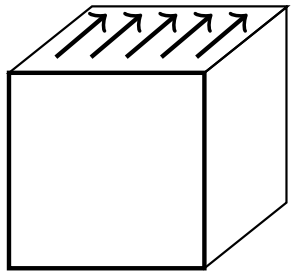
Par séparabilité de la transformée de Fourier, on peut calculer la transformée de Fourier de chaque dimension indépendamment des autres dimensions. On prend une matrice 3D A :

- Calcul de la transformée de Fourier des vecteurs de A dans la dimension x (on obtient la matrice B)
- Calcul de la transformée de Fourier des vecteurs de B dans la dimension y (on obtient la matrice C)
- Calcul de la transformée de Fourier des vecteurs de C dans la dimension z (on obtient la matrice $\hat{A}$)

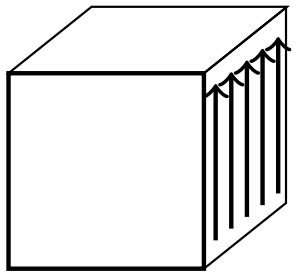
La matrice $\hat{A}$ correspond à la transformée de Fourier discrète en 3D de la matrice A.



(a) Direction X



(b) Direction Y



(c) Direction Z

Multiplication matrice-matrice

Rappel

Le produit de deux matrices $A = (a_{ij})$ de taille $m \times n$ et $B = (b_ij)$ de taille $n \times p$ est la matrice $C = AB$ de taille $m \times p$ telle que :
$c_{ij} = \sum_{k=0}^{n-1} a_{ik} b_{kj}$

Théorème (Complexité du produit de matrices)

La complexité du calcul du produit de deux matrices de taille $N \times M$ et $M \times P$ est en $O(NPM)$.
En particulier, pour deux matrices carrées de taille $N \times N$, la complexité du calcul du produit est en $O(N^3)$.

Démonstration.

Si on calcule le produit de deux matrices de taille $N \times M$ et $M \times P$, on calcule $N \times P$ termes. Chaque terme est calculé par une somme de M termes en effectuant une multiplication pour chaque terme. On a donc, pour chaque terme, M additions et M multiplications. La complexité du calcul est donc en $O(NPM)$. □

Algorithmes de multiplications de matrices

But : obtenir une complexité inférieure à $O(N^3)$.

Algorithme de Strassen (1969)

- Il est possible de multiplier deux matrices 2×2 avec 7 opérations (et non pas 8)
- En appliquant récursivement ce principe on peut multiplier deux matrices $N \times N$.

Complexité : $O(N^{log_2 7}) \simeq O(N^{2,807})$
Défaut : problèmes de stabilité numérique (erreurs d'arrondi dues à la précision finie de la machine).

Algorithme de Coppersmith-Winograd (1987)

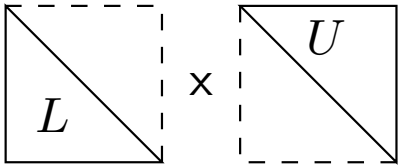
- Complexité en $O(N^{2.376})$
- Attention : la notation O cache une constante : en l'occurrence, elle est très grosse $\rightarrow$ prohibitif sur des matrices qui ne sont pas énormes

Décomposition LU

Définition

La décomposition LU d'une matrice inversible A est sa décomposition en deux matrices L et U avec :

- $A = LU$
- L est une matrice triangulaire inférieure
- U est une matrice triangulaire supérieure



Calcul de la décomposition LU

On la calcule par élimination gaussienne

- On calcule les colonnes une par une
- On détermine un pivot de Gauss
- On élimine les termes sous la diagonale par combinaison linéaire

Théorème (Complexité de la factorisation LU)

La complexité du calcul de la factorisation LU d'une matrice de taille $M \times N$ est en $O(\frac{2}{3}MN^2)$

Décomposition LU : applications

Applications : résolutions de systèmes linéaires, inversion de matrices, calcul de déterminant...

Résolution de système avec LU

On cherche à résoudre le système $Ax = b$.

- On décompose A en $A = LU$
- Le système devient alors $= LUx = b$
- On résout en deux étapes :
  - $Ly = b$
  - $Ux = y$

Stabilité numérique

L'utilisation d'un pivot apporte un risque de propagation des erreurs d'arrondi

- L'utilisation d'un pivot global améliore la stabilité numérique
- Mais la recherche du pivot global augmente le nombre d'opérations à effectuer

Décomposition de Cholesky

Définition

La décomposition de Cholesky d'une matrice A définie positive et symétrique est sa décomposition en une matrice L triangulaire inférieure telle que $A = LL^T$.

Applications : chimie quantique, inversion de matrice
Exemple :

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 5 & 5 & 5 \\ 1 & 5 & 14 & 14 \\ 1 & 5 & 14 & 15 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 \\ 1 & 2 & 3 & 0 \\ 1 & 2 & 3 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 2 & 2 & 2 \\ 0 & 0 & 3 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Théorème (Existence, unicité et complexité)

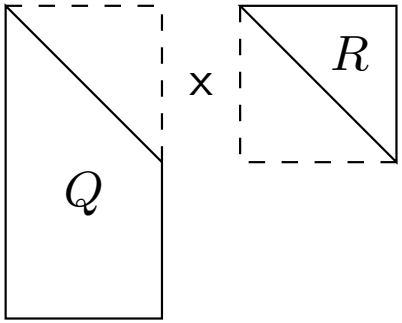
Si A est une matrice carrée symétrique définie positive, alors il existe au moins une matrice réelle triangulaire inférieure L telle que $A = LL^T$. (existence)
Si on impose que tous les éléments diagonaux de la matrice L sont positifs, alors cette matrice L est unique. (unicité)
La complexité du calcul de la décomposition de Cholesky d'une matrice carrée de taille N est en $O(\frac{N^3}{3})$. (complexité)

Factorisation QR

Définition

La décomposition QR (ou factorisation QR) d'une matrice A est sa décomposition en deux matrices Q et R avec :

- $A = QR$
- R est une matrice triangulaire supérieure
- Q est une orthogonale



Applications : résolution de systèmes linéaires non carrés, décomposition en éléments propres, moindres carrés...

Théorème (Existence et unicité)

Si A est une matrice inversible, alors il existe au moins une décomposition QR de A. (existence)
Si on impose que tous les éléments diagonaux de la matrice R sont positifs, alors cette décomposition est unique. (unicité)

Décomposition QR : procédé de Gram-Schmidt

Calcul de la décomposition QR par Gram-Schmidt

En appliquant le procédé de Gram-Schmidt sur les colonnes de la matrice A l'une après l'autre, on effectue des **orthogonalisations** successives sur les vecteurs de la matrice.

Théorème (Complexité de la factorisation QR par le procédé de Gram-Schmidt)

La complexité du calcul de la factorisation QR d'une matrice de taille $M \times N$ en utilisant le procédé de Gram-Schmidt est en $O(MN^2)$

Démonstration.

La matrice A a N colonnes : on doit donc appliquer le procédé de Gram-Schmidt N fois. Chaque orthogonalisation requiert $O(MN)$ opérations. Donc le calcul de la factorisation complète nécessite au total $O(MN^2)$ opérations. □

Décomposition QR : méthode de Householder

Méthode de Householder

On applique des réflecteurs H_k successifs pour projeter un à un les vecteurs de la matrice A sur un hyperplan.

Exemple :

$$A = \begin{bmatrix} x & x & x \\ x & x & x \\ x & x & x \end{bmatrix} \qquad H_1 A = \begin{bmatrix} x & x & x \\ 0 & x & x \\ 0 & x & x \end{bmatrix} \qquad H_2(H_1 A) = \begin{bmatrix} x & x & x \\ 0 & x & x \\ 0 & 0 & x \end{bmatrix}$$

On obtient $R = H_2 H_1 A$, d'où on déduit $Q = H_2 H_1$

Théorème

Le calcul de la factorisation QR d'une matrice de taille $M \times N$ en utilisant l'algorithme de Householder nécessite $2MN^2 - 2/3N^3$ opérations en calculant uniquement la matrice R, et $4MN^2 - 4/3N^3$ opérations en reconstituant également la matrice Q. Sa complexité est donc en $O(MN^2)$.

Décomposition QR : méthode de Givens

Rotations de Givens

Une matrice M représentant une rotation de Givens d'angle θ est de la forme :

$$M = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}$$

Appliquée sur un vecteur x, elle effectue une rotation d'angle θ de ce vecteur.

Méthode de Givens pour la décomposition QR

La méthode de Givens consiste à appliquer des rotations aux vecteurs de la matrice A. La rotation annule les valeurs de A situées sous la diagonale : on obtient ainsi la matrice R. La matrice de rotation résultant de toutes les rotations individuelles est la matrice Q.

Théorème

Le calcul de la factorisation QR d'une matrice de taille $M \times N$ en utilisant la méthode de Givens nécessite $3MN^2 - N^3$ opérations. Sa complexité est donc en $O(MN^2)$.

Cholesky QR

Fonctionnement

On utilise Cholesky pour calculer la factorisation QR d'une matrice A :

- $B = A^T A$
- $R = chol(B)^T$
- $Q = A/R$

Théorème (Complexité de la factorisation QR par Cholesky QR)

La complexité du calcul de la factorisation QR d'une matrice de taille $M \times N$ en utilisant l'algorithme de Cholesky-QR est en $O(MN^2)$ et nécessite $2MN^2 + \frac{N^3}{3}$ pour calculer Q et R (si on ne calcule que R, on a besoin de $MN^2 + \frac{N^3}{3}$).

Démonstration.

La multiplication de matrices $A^T A$ nécessite MN^2 opérations. La factorisation de Cholesky de la matrice B résultante de taille $N \times N$ nécessite $\frac{N^3}{3}$ opérations. S'ajoutent MN^2 opérations supplémentaires pour calculer la matrice Q ($Q = A/R$). □

Routines de base d'algèbre linéaire

Les BLAS

Ce sont des routines qui sont à la base de la plupart des calculs d'algèbre linéaire.

- Interface devenue un standard *de facto*
- Implémentations performantes, spécialisées (liées au matériel) ou généralistes

Calculs en précision simple, double, complexe et double complexe. Remarque : la première lettre du nom de la routine donne la précision (s, d, c, z).

Niveaux de BLAS

Rangées dans 3 catégories : les `niveaux`

- **Niveau 1** : opérations *vectorielles* de type $y = \alpha x + y$, produit scalaire, produit vectoriel, norme...
- **Niveau 2** : opérations *matrice-vecteur* de type $y = \alpha Ax + \beta y$, résolution de $Tx = y$ avec T triangulaire...
- **Niveau 3** : opérations *matrice-matrice* de type $C = \alpha AB + \beta C$, résolution de $C = \alpha T^{-1}C$...

Implémentations

- BLAS de référence (Netlib) : ancien, implémenté en Fortran, pas forcément les plus rapides
- GotoBLAS : implémenté en C, portable, rapide
- BLAS propriétaires : fourni avec la machine (MKL pour Intel, ACML pour AMD, ESSL pour IBM...), optimisé pour une plate-forme donnée
- Pour GPU : Cuda SDK contient une implémentation

Autres :

- xBLAS : BLAS fonctionnant un précision étendue

Exemple d'utilisation de BLAS

DGEMM : multiplication matrice-matrice

Interface C :

```
void dgemm( char *transa, char *transb, int *m, int *n, int *k,
            double *alpha, double *a, int *lda, double *b,
            int *ldb, double *beta, double *c, int *ldc );
```

```
/* Définition des variables etc */
...

/* Allocation des matrices */
A = (double *)malloc(N*N*sizeof(double));
B = (double *)malloc(N*N*sizeof(double));
C = (double *)malloc(N*N*sizeof(double));

/* Initialisation à des valeurs aléatoires */
dlarnv(&IONE, ISEED, &NN, A);
dlarnv(&IONE, ISEED, &NN, B);
dlarnv(&IONE, ISEED, &NN, C);

/* Appel à DGEMM */
dgemm( &NoTranspose, &NoTranspose, &N, &N, &N, &c__1,
       A, &N, B, &N, &c__1, C, &N );
```

Remarques

- Indications passées à BLAS : NoTranspose...
- Passage des paramètres par adresse (interfaçage avec Fortran)
- Matrices déclarées en 1 dimension (Fortran)

Bibliothèque de résolution de systèmes linéaires

Bibliothèque de résolution de systèmes linéaires

But : fournir des implémentations d'opérations matricielles courantes (factorisations QR, LU, Cholesky...), décomposition en éléments propres, en valeurs singulières...
Comme pour les BLAS : l'interface est un standard *de facto*. On appelle cet ensemble de routines **LAPACK**, bien que LAPACK soit avant tout une *implémentation* (de référence) de ces routines.

Système de nommage

Valable également sur les BLAS.

Lettre	Signification
1	Précision (s, d, c, z)
2-3	Type de matrice (GE = générale, OR = orthogonale...)
4-6	Algorithme (QRF, LU...)

Exemples :

- DGELUB = LU sur une matrice générale en double précision
- ZPOTRF = Cholesky sur une matrice définie positive (PO) en complexes double précision

Implémentations des routines de résolution de systèmes

Dépendance

Ces routines appellent les BLAS

Importance d'avoir des BLAS performants !

- LAPACK : bientôt 20 ans, toujours à la pointe
- TLAS : installation avec auto-tunning de LAPACK : l'installeur choisit les meilleurs algorithmes pour la machine;
- FLAME : implémente LAPACK et les BLAS de niveau 3
- PLASMA : algorithmes de nouvelle génération, spécialisée pour les architectures multi-coeurs

Exemple d'utilisation de LAPACK

DGESV : résolution de système linéaire

Interface C :

```
void dgesv( int *n, int *nrhs, double *a, int *lda,
            int *ipiv, double *b, int *ldb, int *info );
```

```
/* Définition des variables etc */
...

/* Allocation des matrices */
A = malloc( N * N * sizeof( double ) );
B = malloc( N * NRHS * sizeof( double ) );

/* Initialisation à des valeurs aléatoires */
dlarnv( &IONE, ISEED, &NN, A );
dlarnv( &IONE, ISEED, &NR, B );

/* Appel de DGESV */
dgesv( &N, &NRHS, A, &LDA, IPIV, B, &LDB, &info );
```

Conclusion

Importance de l'algèbre linéaire en calcul numérique

Nombreuses applications en physique, ingéniérie, finance, traitement du signal...

- Nécessité de maitriser les opérations de calcul matriciel
- Outils puissants pour les sciences (résolution de systèmes, etc)

Besoin d'implémentations efficaces

Tirer parti des capacités de calcul des ordinateurs

- Implémentations efficaces
- Rendues possibles par des algorithmes performants

Pour retrouver cette présentation :

`http://www-lipn.univ-paris13.fr/~coti/cours`