

# Algèbre linéaire

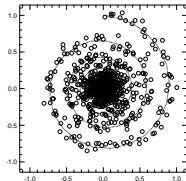
## Applications à la modélisation des données

MNCS – Jean Hare, Noé Lahaye, Jacques Lefrère

Université Pierre et Marie CURIE

### Méthodes numériques pour le calcul scientifique

14 et 15 février 2019



# Sommaire I

- 1 Introduction
  - Notions fondamentales
  - Classification des matrices
  - Norme, conditionnement et spectre des matrices carrées
- 2 Représentation en mémoire et informatique des matrices
- 3 Produit et autres manipulations
- 4 Solution des systèmes linéaires et inversion des matrices
  - Généralités sur l'inversion
  - La méthode de Gauss
  - Exemple de la méthode de Gauss-Jordan
  - Méthodes itératives
- 5 Diagonalisation des matrices
  - Généralités sur la diagonalisation
  - L'algorithme de Jacobi
- 6 Bibliothèques de calcul scientifique

# Sommaire II

- Présentation générale
- Usage de BLAS
- Usage de LAPACK
- Conclusion

## 7 Applications à la modélisation des données

- Analyse du problème

## 8 Ajustement linéaire

- Régression linéaire
- Cas linéaire général
- La qualité de l'ajustement

## 9 Cas Non-Linéaire

- Généralités
- Méthode du gradient conjugué

## 10 Qualité de l'ajustement : cas général

# Algèbre linéaire

- Applications très diverses (physique linéaire ou quasi-linéaire)
- **Optimisation, modélisation**
- **EDP linéaires** (Thème 4)
- Indispensable aussi pour certains calculs *non-linéaires*
- Problèmes spécifiques :
  - très gros volumes de données (ressources en  $N^2$ )
  - matrices creuses, matrices bandes
  - multiplication : en  $N^3$
  - précision après un très grand nombre d'opérations
  - tableaux 2-D en C
- Solutions spécifiques :
  - Procédures de base fortement optimisées (BLAS)
  - **Bibliothèques** diverses (LAPACK)
  - Vectorisation, calcul parallèle (non abordé ici)

LAPACK et BLAS sont intégrés dans Python, Scilab/MatLab, Maple *etc*

# Matrices rectangulaires

$A$  : matrice  $M$  lignes,  $N$  colonnes, ou de profil (*shape*)  $(M, N)$ .

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1j} & \cdots & a_{1N} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{i1} & a_{i2} & \cdots & a_{ij} & \cdots & a_{iN} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{M1} & a_{M2} & \cdots & a_{Mj} & \cdots & a_{MN} \end{pmatrix}$$

**Nota important :** Par convention, en mathématiques,

- $i \equiv$  premier indice  $\equiv$  indice de ligne, de 1 à  $M$
- $j \equiv$  deuxième indice  $\equiv$  indice de colonne, de 1 à  $N$

Matrice carrée : cas particulier  $N = M$ .

Vecteur colonne  $\rightarrow N = 1$ ; vecteur ligne  $\rightarrow M = 1$ .

# Matrices particulières I

**Réelle symétrique** : matrice qui coïncide avec sa transposée  ${}^tA = A$  soit

$$A(i, j) = A(j, i) ;$$

**Hermitienne** matrice complexe qui coïncide avec son adjointe

(auto-adjointe)  $A = A^* = \overline{{}^tA}$ , soit

$$A(i, j) = \text{CONJG}(A(j, i)) ;$$

**Orthogonale** matrice réelle vérifiant  ${}^tA = A^{-1}$  ;

**Unitaire** matrice complexe vérifiant  $A^* = A^{-1}$  ;

**(Symétrique) définie positive** matrice dont les valeurs propres sont strictement positives ;

**À diagonale dominante**  $|A(i, i)| \geq \sum_{\substack{j=1 \\ j \neq i}}^n |A(i, j)| \quad \forall i = 1, 2, \dots, n$

**À diagonale strictement dominante**  $|\dots| > \sum \Rightarrow$  inversible

# Matrices particulières II

**Creuse** matrice contenant beaucoup de zéros...

**Diagonale** matrice dont les éléments non-diagonaux sont nuls

Cas particulier : matrice *scalaire*  $A = \lambda \mathbf{1}$  ;

**Diagonale par bande** seuls ses éléments tels que  $|i - j| \leq p$  ( $p \ll N$ ) sont non-nuls

Cas particulier : matrice *tridiagonale* pour  $p = 1$  ;

**Triangulaire supérieure/inférieure** tous les éléments au-dessous de la diagonale ( $i > j$ ) /au-dessus de la diagonale ( $i < j$ ) sont nuls ;

**Hessenberg** tous les éléments en dessous (au dessus) de la première sous-diagonale (sur-diagonale) sont nuls ;

**Diagonale par blocs** matrice nulle en dehors d'une diagonale de blocs carrés ;

# Norme des matrices carrées

## Exemples de normes des vecteurs

Euclidienne ou  $L_2$   $\|\vec{u}\|_2 = \sqrt{\sum_{i=1}^n u_i^2}$   
 $\|\vec{u}\|_2 \leq 1$  définit une sphère

Sup  $L_\infty$   $\|\vec{u}\|_\infty = \text{Max}_{1 \leq i \leq n} |u_i|$   
 $\|\vec{u}\|_\infty \leq 1$  définit un hypercube

normes équivalentes  $\|\vec{u}\|_\infty \leq \|\vec{u}\|_2 \leq \sqrt{n} \|\vec{u}\|_\infty \quad \forall \vec{u}$

Distance entre deux vecteurs  $\|\vec{u} - \vec{v}\|$

## Normes induites d'une matrice carrée

$$\|A\| = \text{Max}_{\|\vec{u}\|=1} \|A\vec{u}\| = \text{Max}_{\vec{u} \neq 0} \frac{\|A\vec{u}\|}{\|\vec{u}\|}$$

Propriété

$$\|A\|_\infty = \text{Max}_{1 \leq i \leq n} \sum_{j=1}^n |a_{ij}|$$



# Conditionnement des matrices

## Définition

Si  $A$  est inversible, on définit le **conditionnement** de  $A$  par :

$$\gamma(A) = \|A\| \|A^{-1}\| \geq 1$$

## Système mal conditionné

Une matrice « proche » d'une matrice singulière possède un déterminant proche de zéro et un conditionnement élevé.

La résolution du système  $A\vec{x} = \vec{b}$  est alors très sensible aux erreurs d'arrondi. On le qualifie de système « mal conditionné ».

Exemple : calcul du point d'intersection de deux droites presque parallèles.

# Conditionnement et rayon spectral des matrices

## Spectre des matrices carrées

**Spectre d'une matrice** = ensemble de ses valeurs propres

**Rayon spectral** d'une matrice

$$\rho(\mathbf{A}) = \max_i |\lambda_i|$$

où les  $\lambda_i$  sont les valeurs propres de  $\mathbf{A}$ .

Propriété

$$\|\mathbf{A}\|_2 = \sqrt{\rho(\mathbf{A}^t\mathbf{A})} \quad \text{et si } \mathbf{A} \text{ est symétrique} \quad \|\mathbf{A}\|_2 = \rho(\mathbf{A})$$

Donc si  $\mathbf{A}$  est hermitienne, son conditionnement est :

$$\gamma(\mathbf{A}) = \frac{|\lambda|_{\max}}{|\lambda|_{\min}}$$

# Matrices convergentes

## Définition

Matrice  $A$  convergente si

$$\lim_{k \rightarrow \infty} (A^k) = 0$$

## Propriétés

Matrice  $A$  convergente équivaut à

$$\lim_{k \rightarrow \infty} (\|A^k\|) = 0$$

ou à

$$\lim_{k \rightarrow \infty} (A^k \vec{u}) = 0 \quad \forall \vec{u}$$

ou encore à

$$\rho(A) < 1$$

# Stockage des matrices

En Fortran : Vrais Tableaux 2-D,  $A(i, j)$ ,

indexés par défaut de 1 à M et de 1 à N comme les matrices

- statiques, déclarés avec leurs dimensions : `real :: a(3,5)`
- dynamiques, alloués avec `allocate`, libérés avec `deallocate` :  
`real, allocatable :: a(:, :); m=3; n=5; allocate(a(m,n))`
- automatiques (moins utiles qu'en C).

En C : Tableaux de tableaux (pseudo 2-D) :  $A[i][j]$ ,

indexés de 0 à M-1 et 0 à N-1, **pas** comme les matrices.

- statiques, déclarés avec leurs dimensions : `float A[3][5];`
- dynamiques, alloués avec `calloc` (ou `malloc`), libérés avec `free`  
(eg via la bibliothèque `mkitab` : `float ** A; A=float2d(3,5)`)
- automatiques grâce à une **déclaration tardive** en C99 :  
`M=3; N=5; float A[M][N];`

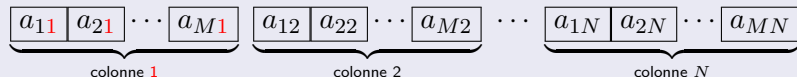
La **portée** des tableaux automatiques est toujours limitée à la procédure.

**NB:** Pour utiliser des tableaux automatiques, réserver suffisamment de place sur la pile (stack) avec : **Linux** : `ulimit -s 524288` (en kbytes).

# Ordre des éléments

## En Fortran

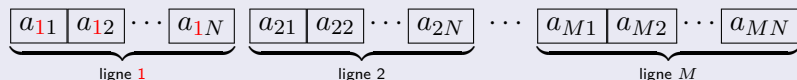
Les éléments de la **même colonne** (second indice) sont contigus :



Indice de **ligne** (1<sup>er</sup> indice) → **indice rapide** : stockage en « **Column Major** ».

## En C

Les éléments de la **même ligne** (premier indice) sont contigus :



Indice de **colonne** (2<sup>nd</sup> indice) → **indice rapide** : stockage en « **Row Major** ».

# Matrice aplatie

Dans les deux langages, on peut aussi utiliser un **tableau 1-D**  $A_p$  pour représenter la **matrice  $A(M,N)$  « aplatie »**. C'est en particulier nécessaire pour l'usage de BLAS et de LAPACK **en langage C** :

- En Fortran, pour accéder à  $a_{ij}$ , remplacer  $A(i,j)$  par  $A_p(i+M*(j-1))$  car **ColMajor** ;  
avancer de **M** cases (le nombre de lignes) quand on incrémente  $j$   
Leading Dimension = **M**
- En C, pour accéder à  $a_{ij}$ , remplacer  $A[i-1][j-1]$  par
  - $A_p[(i-1)+M*(j-1)]$  en **ColMajor**, Leading Dimension = **M**
  - ou  $A_p[N*(i-1)+(j-1)]$  en **RowMajor**, Leading Dimension = **N**

⇒ Écrire une fonction qui calcule l'indice informatique  $k$  dans le tableau aplati de l'élément d'indices mathématiques  $i$  et  $j$  en représentation 2D. Elle utilise la Leading Dimension

# Produit lignes par colonnes

**Définition** : Composition des applications linéaires : si  $A$  est une matrice de profil  $(M, N)$  et  $B$  est une matrice de profil  $(N, P)$ , leur produit est une matrice de profil  $(M, P)$ .

$$C = A.B \iff (C)_{ij} = (A.B)_{ij} = \sum_{k=1}^N A_{ik} B_{kj}$$

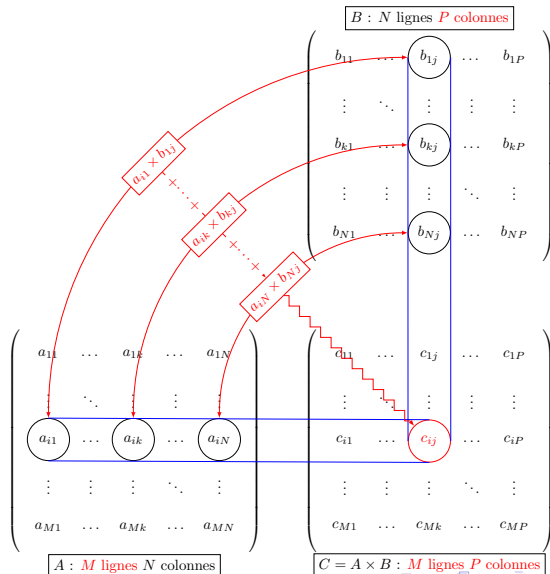
Élément  $C_{ij} = \textit{produit scalaire}$  de la ligne  $i$  de la matrice  $A$  avec la colonne  $j$  de la matrice  $B$ .

**Cas particulier**  $P = 1$  : action de  $A$  sur le vecteur-colonne  $B$ ,  
ou  $M = 1$  : action (à gauche) de la matrice  $B$  sur le vecteur-ligne  $A$ .

**En Fortran** : On utilise la fonction standard :  $C = \text{MATMUL}(A, B)$ .

**En C** : Cette fonction doit être implémentée avec des boucles.

# Multiplication matricielle





# Produit matriciel en C

**Trois** boucles imbriquées : indices extérieurs  $\rightarrow$  indices libres  $i$  et  $j$   
 boucle interne  $\rightarrow$  somme sur  $k$ .

Exemple, matrices aplaties en **RowMajor** (déconseillé!) :

```
void matmul_naive(int M, int N, int P,           // RowMajor
                 float A[M*N], float B[N*P], float C[M*P]) {
    int i,j,k;
    float tmpij;
    for (i=0; i<M; ++i) {
        for (j=0; j<P; ++j) {
            tmpij = 0.0f;
            for (k=0; k<N; ++k) {tmpij += A[i*N+k] * B[k*P+j];}
            C[i*P+j] = tmpij;
        }
    }
}
```

3 boucles imbriquées  $\Rightarrow M \times N \times P \sim N^3$  multiplications  
 et autant d'accès non-contigus dans les tableaux.

# Autres manipulations

Le **Fortran** fournit encore de nombreuses fonctions de manipulation de matrices : `PRODUCT`, `SUM`, `TRANSPOSE`, `RESHAPE`, `DOT_PRODUCT`... enrichies par la notion de section de tableau et l'argument optionnel `DIM`.

En **C** toutes ces fonctions doivent être codées à l'aide de boucles imbriquées.

Pour une meilleure efficacité (vitesse d'exécution), surtout lorsqu'on doit traiter des matrices de taille importante, il devient essentiel d'utiliser des codes soigneusement optimisés. C'est, pour les deux langages, le principal intérêt des bibliothèques mathématiques, et nous présenterons dans la suite de ce cours l'utilisation d'une de ces bibliothèques, le couple `BLAS/LAPACK`.

# Matrices opérant sur des matrices

Pour réaliser les inversions, diagonalisations et factorisations, on modifie de façon graduelle une matrice, par exemple en permutant des lignes et/ou des colonnes, en effectuant des combinaisons linéaires (CL) des lignes et/ou colonnes, *etc.*

Ces manipulations sont des *multiplications* par des matrices creuses.

On notera le résultat qui peut sembler contre-intuitif :

- La multiplication à gauche opère sur les lignes :  
les nouvelles lignes sont des CL des anciennes avec des poids définis par la ligne considérée de la matrice de gauche ;
- La multiplication à droite opère sur les colonnes :  
les nouvelles colonnes sont des CL des anciennes avec des poids définis par la colonne considérée de la matrice de droite ;

Ces règles sont utiles pour décrire les opérations mathématiques, et pour éviter de faire un produit complet (en  $N^3$ ) lorsque la matrice « opérante » est creuse, comme c'est le cas de l'inversion par la méthode de Gauss, ou de la diagonalisation par l'algorithme de Jacobi (*cf infra*).

# Inversion : position du problème

**Définition :** Inverse d'une matrice *carrée*  $A$  de profil  $(N, N) \rightarrow$  matrice  $A^{-1}$  de même dimension telle que  $A.A^{-1} = A^{-1}.A = \mathbf{1}$ .

- permet de résoudre les systèmes linéaires de  $N \times N$
- permet de calculer le déterminant d'une matrice.

Si l'inversion peut sembler plus ardue que la multiplication, sa complexité algorithmique et le nombre d'opérations ne sont pas beaucoup plus grands. Mais elle implique des divisions flottantes, plus longues que des multiplications, et qui peuvent poser des problèmes de précision (matrice mal conditionnée).

Deux types de méthodes pour inverser une matrice :

- méthodes **directes** (pivot de Gauss et ses variantes dont LU)
- méthodes **itératives** (Jacobi, Gauss-Seidel, Successive Over-Relaxation = **SOR**, ...)

# Méthode du pivot de Gauss

**Définition** : Forme systématique de la méthode « d'élimination » ; parcours successif de toutes les lignes.

Étape  $p$  : combinaisons linéaires de lignes  $\rightarrow$  annule tous les éléments de la colonne  $p$  sauf l'élément diagonal  $a_{pp}$  : pivot.

**Matrice singulière** : un pivot(s) nul(s)  $\rightarrow$  matrice singulière (système réduit fournit une équation du sous-espace vectoriel des solutions.)

**Déterminant** = produit des pivots ; méthode en  $N^3$  très économique (par rapport au développement selon une ligne en  $N!$  :  $N = 10 \rightarrow$  gain d'un facteur 3000 environ !)

**Décomposition LU (Lower Upper)** : annuler seulement les éléments situés *sous* la diagonale  $\rightarrow$  matrice triangulaire *supérieure*.

éliminations effectuées en multipliant successivement à gauche par des matrices triangulaires *inférieures*  $\rightarrow$  décomposition LU.

# Gauss-Jordan : principe de base

À l'étape  $n^{\circ}p$  (où  $p = 1, \dots, N$ ) on effectue les opérations suivantes :

- ① éventuellement : recherche de l'élément max dans la colonne  $p$  sur les lignes en dessous de  $p$  ( $q \geq p$ ), et permutation des lignes  $p$  et  $q$
- ② division de la ligne  $p$  par le pivot  $a_{pp} : a_{pp} \leftarrow 1$  ;
- ③  $q \neq p : L_q \leftarrow L_q - a_{qp}L_p$  (soit  $a_{qr} \leftarrow a_{qr} - a_{qp} \times a_{pr}$ )  
 $\Rightarrow a_{pq} = 0$  pour  $q \neq p$ .

Chaque opération sur les lignes = multiplication à gauche de la matrice à inverser  $A$  par une matrice « opérant sur les lignes ».

Ensemble de ces opérations sur  $A \Rightarrow \mathbb{1}_N$

Ensemble de ces opérations sur  $\mathbb{1}_N \Rightarrow B = A^{-1}$

# Un exemple avec pivotage partiel (1/3)

Premier pivot : élément  $m_{11}$  ; formation de la matrice d'élimination  $L_1$  :

$$M = \begin{pmatrix} \mathbf{1} & 2 & 3 & 2 \\ -1 & 2 & -2 & -1 \\ 0 & 3 & -1 & 1 \\ -1 & 3 & -2 & 0 \end{pmatrix} \Rightarrow L_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ \mathbf{1} & 1 & 0 & 0 \\ \mathbf{0} & 0 & 1 & 0 \\ \mathbf{1} & 0 & 0 & 1 \end{pmatrix},$$

dont on déduit les matrices  $M' = L_1 \cdot M$  et  $N' = L_1 \cdot \mathbf{1}$  :

$$M' = L_1 \cdot M = \begin{pmatrix} 1 & 2 & 3 & 2 \\ 0 & 4 & 1 & 1 \\ 0 & 3 & -1 & 1 \\ 0 & \mathbf{5} & 1 & 2 \end{pmatrix}, \quad N' = L_1 \cdot \mathbf{1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

Second pivot :  $m'_{52} = 5 \rightarrow$  ligne 2  $\leftarrow$  matrice de permutation  $S(2, 4)$

## Exemple pivot de Gauss (2/3)

Les matrices  $S(2, 4)$ ,  $S \cdot M'$  et  $L_2$  sont données par :

$$S = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}, \quad S \cdot M' = \begin{pmatrix} 1 & 2 & 3 & 2 \\ 0 & 5 & 1 & 2 \\ 0 & 3 & -1 & 1 \\ 0 & 4 & 1 & 1 \end{pmatrix}, \quad L_2 = \begin{pmatrix} 1 & -2/5 & 0 & 0 \\ 0 & 1/5 & 0 & 0 \\ 0 & -3/5 & 1 & 0 \\ 0 & -4/5 & 0 & 1 \end{pmatrix}$$

d'où  $M'' = L_2 \cdot S(2, 4) \cdot M'$ ,  $N'' = L_2 \cdot S(2, 4) \cdot N'$  :

$$M'' = \begin{pmatrix} 1 & 0 & 13/5 & 6/5 \\ 0 & 1 & 1/5 & 2/5 \\ 0 & 0 & -8/5 & -1/5 \\ 0 & 0 & 1/5 & -3/5 \end{pmatrix}, \quad N'' = \begin{pmatrix} 3/5 & -2/5 & 0 & 0 \\ 1/5 & 1/5 & 0 & 0 \\ -3/5 & -3/5 & 1 & 0 \\ 1/5 & -4/5 & 0 & 1 \end{pmatrix}.$$

Le *troisième pivot* est alors  $m'_{33} = -8/5$  (pas de permutation).



## Exemple pivot de Gauss (3/3)

On obtient alors respectivement  $L_3$ ,  $M''' = L_3 \cdot M''$  et  $N''' = L_3 \cdot N''$  :

$$\begin{pmatrix} 1 & 0 & -13/8 & 0 \\ 0 & 1 & -1/8 & 0 \\ 0 & 0 & -5/8 & 0 \\ 0 & 0 & -1/8 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 7/8 \\ 0 & 1 & 0 & 3/8 \\ 0 & 0 & 1 & 1/8 \\ 0 & 0 & 0 & -5/8 \end{pmatrix}, \begin{pmatrix} -3/8 & 0 & 13/8 & -11/8 \\ 1/8 & 0 & 1/8 & 1/8 \\ 3/8 & 0 & -5/8 & 3/8 \\ 1/8 & 1 & 1/8 & -7/8 \end{pmatrix}.$$

Le *quatrième pivot* est alors  $m'_{44} = -5/8$ , donc  $L_4$ ,  $M'''' = \mathbf{1}$ , et  $N'''' \equiv M^{-1}$  sont

$$\begin{pmatrix} 1 & 0 & 0 & -7/5 \\ 0 & 1 & 0 & -3/5 \\ 0 & 0 & 1 & -1/5 \\ 0 & 0 & 0 & -8/5 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \text{ et } M^{-1} = \frac{1}{5} \begin{pmatrix} -1 & 7 & 9 & -13 \\ 1 & 3 & 1 & -2 \\ 2 & 1 & -3 & 1 \\ -1 & -8 & -1 & 7 \end{pmatrix}.$$

Le déterminant est le produit des pivots :  $\det(M) = 1 \times 5 \times \frac{-8}{5} \times \frac{-5}{8} = 5$  comme un calcul plus ordinaire mais plus laborieux nous l'aurait donné.

# Décompositions usuelles

Bien des opérations reposent sur différentes factorisations de la matrice, dont les facteurs se prêtent à une inversion simple :

- **Factorisation LU** :  $A = P \cdot L \cdot U$ , où
  - $P$  : matrice de permutation (cf exemple 2 ci-dessus)
  - $L$  (lower) : triangulaire inférieure, éléments diagonaux égaux à 1,
  - $U$  (upper) : triangulaire supérieure.

S'obtient par une modification minime, mais efficace de la méthode de Gauss.

- **Factorisation de Cholesky** :  $A = L \cdot L^*$  ou  $A = U^* \cdot U$ , et autres variantes ( $A$  symétrique définie positive).
- **Factorisation QR** :  $A = Q \cdot R$  où  $Q$  : matrice orthogonale (ou unitaire si  $A$  est complexe) et  $R$  : matrice triangulaire supérieure.

et bien d'autres méthodes fondées sur la décomposition en valeurs singulières, ou la diagonalisation.

# Introduction aux méthodes itératives

## Principe

**Itérer** une application affine de matrice  $M$  et de vecteur de translation  $C$  sur le vecteur  $X$  :

$$X^{(k)} = MX^{(k-1)} + C$$

telle que le **point fixe** soit solution du système  $AX^{(\infty)} = B$

**Critère d'arrêt** :  $\|X^{(k)} - X^{(k-1)}\| \leq \eta \|X^{(k-1)}\|$

Série géométrique de raison  $M^k$  : **Convergence si  $\rho(M) < 1$**

Les méthodes se distinguent par le choix de la matrice  $M$  et du vecteur  $C$ .

La matrice  $A$  du système est décomposée en  $A = D + L + U$

- $D$  diagonale,
- $L$  triangulaire **inférieure** (stricte),
- $U$  triangulaire **supérieure** (stricte).

$D$  est supposée inversible ( $a_{ii} \neq 0 \forall i$ ), donc  $D + L$  est inversible.

# Méthode de Jacobi

En utilisant la  $i^e$  équation du système  $AX = B$ ,

$$a_{i1}x_1 + \cdots + a_{i,i-1}x_{i-1} + a_{ii}\boxed{x_i} + a_{i,i+1}x_{i+1} + \cdots + a_{in}x_n = b_i$$

calculer chaque composante  $i$  du nouveau vecteur  $X^{(k)}$  en fonction des autres composantes ( $j \neq i$ ) du vecteur antérieur  $X^{(k-1)}$ .

$$x_i^{(k)} = -\frac{1}{a_{ii}} \sum_{j=1}^{j=i-1} a_{ij}x_j^{(k-1)} - \frac{1}{a_{ii}} \sum_{j=i+1}^{j=n} a_{ij}x_j^{(k-1)} + \frac{b_i}{a_{ii}} \quad (1)$$

$$X^{(k)} = -D^{-1}(L + U)X^{(k-1)} + D^{-1}B$$

$$M_J = -D^{-1}(L + U) \quad \text{et} \quad C = D^{-1}B$$

On vérifie que  $X^{(\infty)} = -D^{-1}(L + U)X^{(\infty)} + D^{-1}B$  implique  
 $DX^{(\infty)} = -(L + U)X^{(\infty)} + B$ , soit  $AX^{(\infty)} = B$

# Méthode de Gauss-Seidel

La variante de Gauss-Seidel consiste à utiliser, dans la  $i^e$  équation, les composantes **déjà calculées** ( $j < i$ ) **du nouveau** vecteur pour déterminer la composante  $i$  du nouveau vecteur.

$$x_i^{(k)} = -\frac{1}{a_{ii}} \sum_{j=1}^{j=i-1} a_{ij} x_j^{(k)} - \frac{1}{a_{ii}} \sum_{j=i+1}^{j=n} a_{ij} x_j^{(k-1)} + \frac{b_i}{a_{ii}} \quad (2)$$

$$X^{(k)} = -(D + L)^{-1} U X^{(k-1)} + (D + L)^{-1} B$$

$$M_G = -(D + L)^{-1} U \quad \text{et} \quad C = (D + L)^{-1} B$$

Convergence plus rapide que Jacobi car  $\rho(M_G) = \rho^2(M_J)$

On vérifie que  $X^{(\infty)} = -(D + L)^{-1} U X^{(\infty)} + (D + L)^{-1} B$  implique

$$(D + L)X^{(\infty)} = -UX^{(\infty)} + B, \text{ soit } AX^{(\infty)} = B$$

# Diagonalisation : Position du problème

**Définition** :  $A \rightarrow A = VDV^{-1}$  ( $A$  matrice carrée,  $D$  matrice diagonale et  $V$  matrice inversible, dite « matrice de passage ».)

**Éléments spectraux** Éléments diagonaux de  $D \rightarrow$  valeurs propres de  $A$  ;  
colonnes de  $V \rightarrow$  vecteurs propres correspondants.

**Existence** Cette décomposition n'existe pas toujours, mais les matrices réelles symétriques et complexes hermitiques sont diagonalisables.

Les matrices définies positives (ou négatives) sont toujours diagonalisables avec une matrice de passage qui est orthogonale réelle ou unitaire dans le cas complexe.

**Décomposition de Schur** Cas général : matrice non diagonalisable, on se contente de cette décomposition :  $A = U \cdot T \cdot U^*$

( $U$  unitaire,  $T$  triangulaire supérieure avec les valeurs propres de  $A$  sur la diagonale. Cas particulier : forme canonique de Jordan.)

# L'algorithme de Jacobi

**Algorithme de Jacobi** : méthode de diagonalisation élémentaire des matrices symétriques.

Méthode **itérative** : annule successivement les éléments non-diagonaux

**En dimension 2**, on peut retrancher à la matrice  $A$  à diagonaliser la matrice scalaire  $\text{tr}(A)\mathbb{1}/2$ , sans changer ses vecteurs propres. La matrice  $A'$  ainsi obtenue est proportionnelle à une matrice de symétrie

$S(\theta) = \begin{pmatrix} \cos 2\theta & \sin 2\theta \\ \sin 2\theta & -\cos 2\theta \end{pmatrix}$ , que l'on sait diagonaliser en utilisant la matrice de rotation d'angle  $\theta$ .

**En dimension  $n$**  On utilise la même approche, en annulant successivement les éléments non diagonaux  $a_{pq}$ . En considérant la sous-matrice

$A_{pq} = \begin{pmatrix} a_{pp} & a_{pq} \\ a_{qp} & a_{qq} \end{pmatrix}$ , on peut construire comme en dimension 2 une rotation d'angle  $\theta$  qui n'affecte que les vecteurs de base  $X_p$  et  $X_q$ .

# Algorithme de Jacobi en dimension $n$

A chaque étape, on doit ainsi multiplier  $A$  à gauche par la matrice  $R_{pq}$  ci-dessous, et à droite par sa transposée, avec les notations précédentes, en posant  $c = \cos \theta$  et  $s = \sin \theta$  :

$$A = \begin{matrix} & \begin{matrix} 1 \cdots & p & \cdots & q & \cdots n \end{matrix} \\ \begin{matrix} 1 \\ \vdots \\ p \\ \vdots \\ q \\ \vdots \\ n \end{matrix} & \left( \begin{array}{c|c|c|c|c} & & & & \\ & & & & \\ & & & & \\ \hline & a_{pp} & & a_{pq} & \\ \hline & & & & \\ & & & & \\ \hline & a_{qp} & & a_{qq} & \\ \hline & & & & \\ & & & & \\ \hline & & & & \end{array} \right) \end{matrix} \Rightarrow R_{pq} = \begin{matrix} & \begin{matrix} 1 \cdots & p & \cdots & q & \cdots n \end{matrix} \\ \begin{matrix} 1 \\ \vdots \\ p \\ \vdots \\ q \\ \vdots \\ n \end{matrix} & \left( \begin{array}{c|c|c|c|c} \mathbb{1} & 0 & \mathbb{0} & 0 & \mathbb{0} \\ & \vdots & \mathbb{0} & \vdots & \mathbb{0} \\ \hline 0 \cdots & \mathbf{c} & \cdots & \mathbf{s} & \cdots 0 \\ \hline \mathbb{0} & \vdots & \mathbb{1} & \vdots & \mathbb{0} \\ \hline 0 \cdots & \mathbf{-s} & \cdots & \mathbf{c} & \cdots 0 \\ \hline \mathbb{0} & \vdots & \mathbb{0} & \vdots & \mathbb{1} \\ & 0 & & 0 & \end{array} \right) \end{matrix}$$

Bien sûr on ne calcule pas les produits de matrices, mais simplement :

- 2 CL de lignes pour le produit à gauche
- 2 CL de colonnes pour le produit à droite.



# Itérations et convergence

- Cette opération ne fait qu'annuler deux éléments non-diagonaux : on doit l'itérer, par exemple en choisissant à chaque pas l'élément non-diagonal le plus grand (en valeur absolue).
- On peut montrer que le **caractère non-diagonal**,  $\mathcal{E}(A) = 2 \sum_{i < j} a_{ij}^2$  diminue de  $2a_{pq}^2$  à chaque itération, et en déduire que  $\mathcal{E}$  tend vers zéro plus vite qu'une suite géométrique de raison  $\rho < 1$ , dépendant bien sûr de la matrice A.
- Le processus est donc convergent, on l'arrête en utilisant un *critère de convergence* défini par  $\mathcal{E}(A) \leq \eta^2 \mathcal{N}(A)$  où  $\mathcal{N}(A) = \sum_{i,j} a_{ij}^2$ , et la tolérance  $\eta$  est choisie en fonction de la précision recherchée, mais aussi de façon à éviter d'accumuler des erreurs d'arrondi.
- En définitive, la matrice A est devenue quasi-diagonale, et la matrice de passage est le produit des rotations utilisées.

# Bibliothèques : présentation

- Les codes fondés sur des algorithmes et des procédures d'optimisation bien établis sont « consolidés » dans des bibliothèques.
- Il existe un nombre considérable de bibliothèques, dédiées à de très nombreux types de calcul : AL, FFT et ondelettes, analyse spectrale, intégration, EDO, EDP et éléments finis, fonctions spéciales, traitement d'images, CAO dans différents domaines etc.
- On en recense près de 80 sur l'algèbre linéaire  
<http://www.netlib.org/utk/people/JackDongarra/la-sw.html>
- Chaque fabricant de processeurs ou éditeur de compilateurs orientés « calcul scientifique » propose, moyennant finances, une bibliothèque mathématique qui intègre bien souvent des bibliothèques « open source », optimisées pour leur compilateur ou leurs microprocesseurs.

Voir [http://en.wikipedia.org/wiki/List\\_of\\_numerical\\_analysis\\_software](http://en.wikipedia.org/wiki/List_of_numerical_analysis_software)

# Le couple BLAS/LAPACK

Choix d'une bibliothèque (*library*) du domaine public :

- la bibliothèque **LAPACK** (**Linear Algebra PACKage**), fondée elle-même
- sur la librairie **BLAS** (**Basic Linear Algebra Subprograms**), qui fournit les manipulations élémentaires « de bas niveau ».

Avantages de ces bibliothèques :

- gratuites  $\Rightarrow$  intégrées dans de très nombreux programmes comme Python, Maple, Scilab, Matlab, Octave *etc*
- aussi des versions propriétaires de Intel, Sun, HP, NEC, Irix, NAG, *etc*
- Écrites au départ en **Fortran 77**, elles ont été adaptées à de nombreux langages : Fortran 95, C, C++, Java, Python *etc*
- Un programme fondé sur LAPACK est aisément optimisé pour une nouvelle architecture (vectorisation, calcul parallèle, calcul distribué) en changeant uniquement la version de BLAS sous-jacente.
- Il existe aussi des extensions pour des types de matrices non pris en compte (matrices creuses ou par blocs *etc*).

# Appels en Fortran 95 et en C

**Mise en garde** : pour économiser la mémoire, les procédures travaillent souvent « **en place** », en écrivant le résultat à la place des données d'entrée. Les bibliothèques écrites en Fortran 77 sont directement utilisables depuis le Fortran 95 avec tous ses avantages grâce à des « wrappers » :

- Le module **blas95** sécurise l'appel des sous-routines de BLAS en fournissant leur interface et simplifie les passages de tableaux ;
- Le package **lapack95** simplifie l'appel des sous-routines de LAPACK, en utilisant de plus la généralité et les paramètres optionnels.

Leur utilisation en C rencontre plusieurs difficultés :

- l'inter-opération du C et du Fortran (types de variables, passage des paramètres, absence de généralité *etc*) ;
- l'absence de vrais tableaux 2-D au sens où ils existent en Fortran ;
- l'ordre de rangement des éléments dans les tableaux de tableaux ( $A[i][j]$ ) diffère de celui du Fortran.

# Le problème spécifique du C

La solution aux problèmes spécifiques du C passe par :

- la représentation des matrices dans leur forme **aplatie (1-D)** ;
- L'ajout, dans toutes les fonctions d'un paramètre `CblasRowMajor/RowMajor` ou `CblasColumnMajor/ColumnMajor` indiquant dans quel **ordre** sont rangés les éléments ;
- Une version de BLAS réécrite en C, **OpenBlas**, qui est de plus optimisée automatiquement à la compilation ;
- Une interface dédiée pour le langage C dénommée **lapacke** bien que moins puissante que celle du Fortran 95, elle décharge l'utilisateur de certaines tâches de bas niveau.

Dans le cas où on l'on utilise `RowMajor` – **à éviter** – `lapacke` se charge des transpositions nécessaires avec un algorithme optimisé.

Notons **qu'il faut utiliser** soit des tableaux dynamiques 1D, soit les tableaux automatiques du C99, pour que les éléments soient contigus en mémoire.

# BLAS

BLAS est structuré en trois niveaux, selon la nature des opérations :

- ① vectorielles du type :  $y \leftarrow \alpha x + y$ , ou  $x \cdot y$ , etc
- ② matrices-vecteurs du type :  $y \leftarrow \alpha A.x + \beta y$  et les résolutions de systèmes ;
- ③ matrices-matrices du type :  $C \leftarrow \alpha A.B + \beta C$ , les inversions etc

Le nom des fonctions comme `sgeemm` par exemple est formé par **S-GE-MM** où les lettres désignent :

- ① le **type de données** : **S** = réel simple précision, **D** =double précision, **C** : complexe simple, **Z** complexe double
- ② le **type de matrice** concerné : **GE**=general, **GB**=general band, **SY**(**SB**/**SP**)=symmetric (band/packed), **HE**(**HB**/**HP**)=hermitian (band/packed), **TR**(**TB**/**TP**)=triangular (band/packed)
- ③ La nature de l'**opération** : **AXPY**=somme de vecteurs, **MV**=produit matrice-vecteur, **MM**=produit de matrices, **SV**/**SM** résolution matrice-vecteur/matrice-matrice (**S** comme solve).

# Exemple de multiplication avec GEMM en fortran 95

En fortran 77, l'interface de SGEMM comporte 13 paramètres :

SGEMM(TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)

réalise l'opération  $C \leftarrow \alpha A \cdot B + \beta C$ , où

- TRANSA et TRANSB (des caractères prenant la valeurs 'N' ou 'T') indiquent s'il faut travailler sur A et B ou leur transposée ;
- M, N, K sont les tailles des matrices A(M,N), B(N,K), C(M,K),
- LDA, LDB, LDC sont aussi des tailles :

LD=leading dimension=nbre de lignes car ColMajor en fortran

En fortran 95 avec GEMM, seuls les 3 paramètres A, B et C sont obligatoires.

```

use la_precision, only : WP=>SP           ! la precision
use blas95, only : gemm                   ! la fonction
integer, parameter :: m=...,n=..., p=..  ! les tailles
real(kind=WP) :: A(m,n), B(n,p), C(m,p) ! les matrices
! ... remplissage des matrices A et B ...
call GEMM(A, B, C)                        ! générique sans les étendues
! en F77 : call sgemm('N','N',M,P,N,1.0,A,M,B,N,0.0,C,M)

```

# Exemple de multiplication avec xGEMM

En C : on retrouve les difficultés habituelles, plus celles qui tiennent à ce que la bibliothèque est en Fortran, notamment :

- Il faut utiliser la représentation **aplatie**,
- La « Leading Dimension » dépend de l'ordre choisi pour les éléments : ce n'est pas toujours le nombre de lignes, mais la taille de l'enregistrement de base dans la matrice aplatie. En `CblasRowMajor`, c'est donc la longueur d'une ligne, soit le nombre de colonnes !  
⇒ **Préférer `CblasColMajor`** où Leading Dimension = nbre de lignes

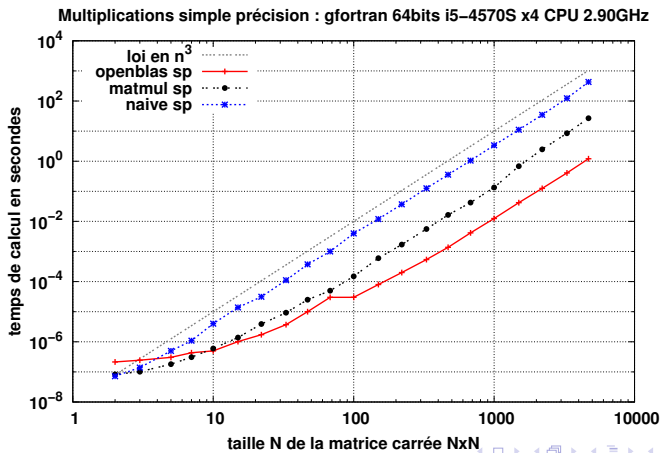
En `CblasRowMajor`, déconseillé, on écrirait par exemple :

```
#include "cblas.h"           // en RowMajor !
int M=.., N=.., P=..;      // les tailles
float A[M*N], B[N*P], C[M*P]; // matrices aplaties
// ... remplissage des matrices A et B ...
cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
            M, P, N, 1.0F, A, N, B, P, 0.0F, C, P);
```



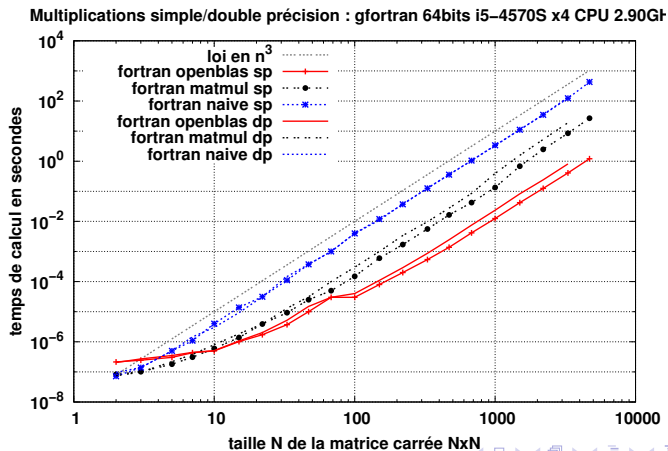
# Gain de temps sur la multiplication avec MATMUL et GEMM

Temps de calcul mesuré sur un I5 (R) quad-core à 2.9 GHz sous linux 64 bits, pour le produit de 2 matrices aléatoires  $N \times N$  en simple précision, via le produit « naïf », avec MATMUL et avec GEMM de openblas en fortran.



# Gains simple et double précision avec MATMUL et xGEMM

Temps de calcul pour le produit de 2 matrices aléatoires  $N \times N$  en simple et double précision, à l'aide du produit « naïf », avec MATMUL et avec GEMM de **openblas** en fortran. Le C donne les mêmes performances.



# LAPACK : principe

LAPACK offre des procédures de plus haut niveau en combinant les possibilités offertes par BLAS.

Les fonctions de LAPACK sont elles-mêmes de 2 niveaux :

- les procédures de **calcul** (computational),
- et les procédures **pilotes** (driver), qui utilisent les précédentes.

Par exemple

- la procédure **pilote** `xGESV(N, NRHS, A, LDA, IPIV, B, LDB, INFO)` résout le système de  $N$  équations à  $N$  inconnues  $A.X = B$ , où l'inconnue  $X$  et le second membre  $B$  ont  $NRHS$  colonnes ;
- `xGESV` appelle les procédures de **calcul** `xGETRF` pour la factorisation LU de  $A$ , puis `xGETRS` pour résoudre le système triangulaire ;
- chacune des deux procédures `xGETRF` elles-mêmes, utilisent les procédures LAPACK `xGETF2` et `xLASWP` ainsi que la procédure BLAS `xTRSM`.

# LAPACK : exemple de résolution de système

En Fortran :

```

use la_precision, only: WP=>SP      ! precision
use F95_LAPACK, only: LA_GESV      ! procedure
integer, parameter :: m=.., p=..  ! tailles
real(WP):: A(m,m), B(m,p)         ! matrice, seconds membres
integer :: info, ipiv(m)          ! erreurs,permutations
! ... remplissage de A et B...
call LA_GESV(A, B, INFO=info, IPIV=ipiv)

```

En C :

```

#include "lapacke.h"
int m=.., p=.. ;                // les tailles
float A[m*m], B[m*p] ;          // matrice, seconds membres
int info, ipiv[m];              // erreurs, permutations
// ... remplissage de A et B ...
info = LAPACKE_sgesv(CblasRowMajor, M, P, A, M, ipiv, B, M);

```

**Attention :** A remplacée par sa décomposition LU et B par les solutions !

# Conclusion : intérêt des bibliothèques

L'utilisation des bibliothèques permet :

- d'écrire du **code compact** en s'appuyant sur des méthodes générales éprouvées, et de bénéficier d'une optimisation logicielle et matérielle maximale, pour se concentrer sur ce qui est spécifique,
- de choisir parmi tout un arsenal la méthode **la plus adaptée**, notamment **selon le type de la matrice** (bandes, triangulaires, symétriques ou hermitiennes) : nous n'avons utilisé dans les exemples que les procédures générales **xGEyy**, mais selon le problème considéré on pourra faire un choix optimal.

Ainsi, pour diagonaliser la matrice  $L_{1D}$  du TE, on peut choisir :

- soit la méthode générale **xGEEV**,
- soit **xSYEV** qui exploite son caractère **symétrique**,
- soit **xSTEV** utilisant son caractère **symétrique et tridiagonal**.

# Optimisation et ajustement

**Optimisation** : thématique transversale des mathématiques.

Applications dans toutes les disciplines :

- Sciences
- Économie
- Recherche opérationnelle (contraintes techniques, coûts variables/incompressibles  $\Rightarrow$  fixer un **ensemble de paramètres** pour maximiser un bénéfice sur divers produits.)

Dans ce cours nous nous limiterons au problème physique des **ajustements de données** expérimentales par des fonctions appelées « **modèle** », et utilisant les méthodes de l'algèbre linéaire.

# Position du problème : ajustement

## Exemple d'ajustement :

Ensemble de  $N$  points « expérimentaux »  $(x_i, y_i, \sigma_i)_{i=1, N}$ .

Barre d'erreur sur  $y_i$  = déviation standard  $\sigma_i$

Mais les  $x_i$  supposés connus exactement

**Objectif** : vérifier que les mesures suivent une loi physique et déterminer au mieux les  $M$  paramètres  $(p_j)_{j=1, M}$  qui caractérisent cette loi.

On suppose que les points devraient vérifier le **modèle**  $f$  :

$$y(x) = f(x, p_1, \dots, p_M) = f(x, \vec{p}) , \quad (3)$$

$\vec{p}$  : vecteur à  $M$  composantes des paramètres.

**NB** : La fonction modèle  $f$  est souvent notée  $y_{\text{th}}$ , voire  $y$  tout court.

# Équations sur-déterminées

$N$  mesures  $\Rightarrow$  déterminations approchées des paramètres  $\vec{p}$  du modèle.

**Problème** :  $N$  équations v.s.  $M$  paramètres avec  $N \gg M$  (en général)

$\Rightarrow$  Équations **sur-déterminées** et souvent **incompatibles**.



# Équations sur-déterminées

$N$  mesures  $\Rightarrow$  déterminations approchées des paramètres  $\vec{p}$  du modèle.

**Problème** :  $N$  équations v.s.  $M$  paramètres avec  $N \gg M$  (en général)  
 $\Rightarrow$  Équations **sur-déterminées** et souvent **incompatibles**.

Exemple : cas **linéaire**  $\rightarrow$  résolution d'un système de  $N$  équations à  $M$  inconnues de la forme :  $A \cdot p = b$  où  $A =$  matrice  $N \times M$ .

En général : pas de solution !

# Équations sur-déterminées

$N$  mesures  $\Rightarrow$  déterminations approchées des paramètres  $\vec{p}$  du modèle.

**Problème** :  $N$  équations v.s.  $M$  paramètres avec  $N \gg M$  (en général)  
 $\Rightarrow$  Équations **sur-déterminées** et souvent **incompatibles**.

Exemple : cas **linéaire**  $\rightarrow$  résolution d'un système de  $N$  équations à  $M$  inconnues de la forme :  $A \cdot p = b$  où  $A =$  matrice  $N \times M$ .

En général : pas de solution !

$\Rightarrow$  **minimiser** la norme  $\|A \cdot p - b\|$  (la norme dépend de la loi),  
par des méthodes directes ou itératives selon la nature du problème.

Approche justifiée par une **analyse statistique** du problème :

$\Rightarrow$  **estimation** des paramètres à partir des mesures.

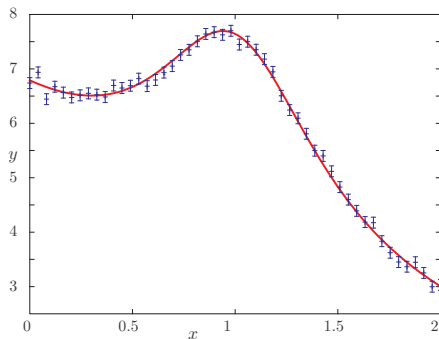
# Déviations

Typiquement, on suppose qu'il existe un jeu de valeurs  $\vec{p}^v$  des paramètres tel que :

$$y_i = f(x_i, \vec{p}^v) + \varepsilon_i \quad (4)$$

où les **déviations**  $\varepsilon_i$  résultent

- de bruit sur les signaux,
- d'un léger écart au modèle (effets non pris en compte),
- des incertitudes introduites dans le processus de mesure.



**NB :** Les abscisses de mesure  $x_i$  sont supposées connues parfaitement.

# Probabilité conditionnelle et vraisemblance

Hypothèses : déviations  $\varepsilon_i$  sur les  $y_i =$  variables aléatoires

- **indépendantes**
- **gaussiennes centrées**, d'écart-type  $\sigma_i$ .

Probabilité conditionnelle des mesures  $y_i$  connaissant les paramètres :

$$\mathcal{P}(y_1, \dots, y_N | \vec{p}) = \prod_{i=1}^N \mathcal{P}(y_i | \vec{p}) \propto \exp \left[ -\frac{1}{2} \sum_{i=1}^N \left( \frac{y_i - f(x_i, \vec{p})}{\sigma_i} \right)^2 \right].$$

# Probabilité conditionnelle et vraisemblance

Hypothèses : déviations  $\varepsilon_i$  sur les  $y_i =$  variables aléatoires

- **indépendantes**
- **gaussiennes centrées**, d'écart-type  $\sigma_i$ .

Probabilité conditionnelle des mesures  $y_i$  connaissant les paramètres :

$$\mathcal{P}(y_1, \dots, y_N | \vec{p}) = \prod_{i=1}^N \mathcal{P}(y_i | \vec{p}) \propto \exp \left[ -\frac{1}{2} \sum_{i=1}^N \left( \frac{y_i - f(x_i, \vec{p})}{\sigma_i} \right)^2 \right].$$

Probabilité conditionnelle des paramètres connaissant les mesures =  
fonction de **vraisemblance** (*likelihood*) :  $\mathcal{L}(\vec{p} | y_1, \dots, y_N)$

Problème d'estimation des paramètres  $\vec{p}$  au vu des mesures :

⇒ choisir l'estimateur **du maximum de vraisemblance**

$$\text{Max} [\mathcal{L}(\vec{p} | y_1, \dots, y_N)]$$

# Probabilité conditionnelle et vraisemblance

Hypothèses : déviations  $\varepsilon_i$  sur les  $y_i =$  variables aléatoires

- **indépendantes**
- **gaussiennes centrées**, d'écart-type  $\sigma_i$ .

Probabilité conditionnelle des mesures  $y_i$  connaissant les paramètres :

$$\mathcal{P}(y_1, \dots, y_N | \vec{p}) = \prod_{i=1}^N \mathcal{P}(y_i | \vec{p}) \propto \exp \left[ -\frac{1}{2} \sum_{i=1}^N \left( \frac{y_i - f(x_i, \vec{p})}{\sigma_i} \right)^2 \right].$$

Probabilité conditionnelle des paramètres connaissant les mesures =  
fonction de **vraisemblance** (*likelihood*) :  $\mathcal{L}(\vec{p} | y_1, \dots, y_N)$

Problème d'estimation des paramètres  $\vec{p}$  au vu des mesures :

⇒ choisir l'estimateur **du maximum de vraisemblance**

$$\text{Max} [\mathcal{L}(\vec{p} | y_1, \dots, y_N)]$$

# Vraisemblance maximale et erreur quadratique minimale

On admet que

$$\text{Max} [\mathcal{L}(\vec{p} | y_1, \dots, y_N)] \iff \text{Max} [\mathcal{P}(y_1, \dots, y_N | \vec{p})]$$

Justifié lorsque la fonction  $f$  est linéaire par rapport aux variables  $\vec{p}$ , (et il n'est alors pas nécessaire que les  $\varepsilon_i$  suivent une loi normale).

Si on considère une distribution a priori des paramètres uniforme, le théorème de Bayes justifie cette identification :

$$\mathcal{P}(\vec{p} | y_1, \dots, y_N) = \mathcal{P}(y_1, \dots, y_N | \vec{p}) \times \mathcal{P}(\vec{p}) / \mathcal{P}(y_1, \dots, y_N)$$

⇒ Minimiser l'erreur quadratique moyenne pondérée :

$$\text{Min} \left[ \sum_{i=1}^N \left( \frac{y_i - f(x_i, \vec{p})}{\sigma_i} \right)^2 \right].$$

# Moindres carrés pondérés

Les meilleures valeurs de  $\vec{p}$ , notées  $\vec{p}^a$ , **maximisent** la vraisemblance  $\mathcal{L}$ , soit encore **minimisent** l'erreur quadratique pondérée totale :

$$\mathcal{E}(\vec{p}) = \sum_{i=1}^N \left( \frac{\varepsilon_i}{\sigma_i} \right)^2 = \sum_{i=1}^N \left( \frac{y_i - f(x_i, \vec{p})}{\sigma_i} \right)^2 . \quad (5)$$

- Dans l'hypothèse où les déviations sont des variables normales centrées, elle a la structure d'un  $\chi^2$ .
- Elle est très souvent abusivement appelée « chi2 » ( $\chi^2$ ), alors qu'elle n'y correspond – au mieux – qu'au point  $\vec{p}^a$  pour lequel on suppose que la valeur moyenne des déviations normalisées est nulle.



# Équations normales

Minimum recherché  $\vec{p}^a$  : point où le gradient de  $\mathcal{E}$  s'annule.

⇒  $M$  **équations normales** :

$$\sum_{i=1}^N \left( \frac{y_i - f(x_i, \vec{p}^a)}{\sigma_i^2} \right) \frac{\partial f(x_i, \vec{p}^a)}{\partial p_k} = 0, \quad k = 1, \dots, M \quad (6)$$

Solution des équations normales = valeurs de  $\vec{p}^a$  qui, **si le modèle est valide**, approchent la « vraie » valeur  $\vec{p}^v$ .

« Ajustement » des données par les moindres carrés pondérés  
ou “weighted least squares fit”

# Linéaire vs non-linéaire

Deux situations à distinguer :

- 1 Cas particulier : fonction  $f(x, \vec{p})$  **linéaire vis à vis des paramètres**  $p_j$ .  
 $\Rightarrow \frac{\partial f(x_i, \vec{p}^a)}{\partial p_k} = \text{cste}$  (par rapport à  $\vec{p}$ )  
système des équations normales **linéaire**  $\rightarrow$  résolution simple.
- 2 Cas général :  $f(x, \vec{p})$  est une fonction **non-linéaire des paramètres**  $p_j$ .  
 $\Rightarrow \frac{\partial f(x_i, \vec{p}^a)}{\partial p_k}$  dépend des  $p_j$ .  
Le système des équations normales est **non-linéaire**  
 $\rightarrow$  méthode **itérative** de résolution.

**NB** : ne pas confondre avec linéarité de  $f$  en fonction de  $x$

# Régression linéaire : équations normales

Cas le plus simple et le plus important : modèle linéaire vis à vis des ( $M = 2$ ) paramètres et aussi des données :

$$f(x, \vec{p}) = a + b x \quad \text{où} \quad \vec{p} = (a, b) \quad (7)$$

Alors

$$\cancel{\chi^2(\vec{p})} = \mathcal{E}(\vec{p}) = \sum_{i=1}^N \left( \frac{y_i - a - b x_i}{\sigma_i} \right)^2 \quad (8)$$

$$\text{Min} [\mathcal{E}] \Rightarrow \begin{cases} \frac{\partial \mathcal{E}}{\partial a} = 0 \Rightarrow \sum_{i=1}^N \frac{y_i - a - b x_i}{\sigma_i^2} = 0 \\ \frac{\partial \mathcal{E}}{\partial b} = 0 \Rightarrow \sum_{i=1}^N x_i \frac{y_i - a - b x_i}{\sigma_i^2} = 0 \end{cases} \quad (9)$$

# Régression linéaire : résolution

En développant, on définit les 6 sommes

$$\begin{aligned}
 S &= \sum_{i=1}^N \frac{1}{\sigma_i^2} & S_x &= \sum_{i=1}^N \frac{x_i}{\sigma_i^2} & S_y &= \sum_{i=1}^N \frac{y_i}{\sigma_i^2} \\
 S_{xy} &= \sum_{i=1}^N \frac{x_i y_i}{\sigma_i^2} & S_{xx} &= \sum_{i=1}^N \frac{x_i^2}{\sigma_i^2} & S_{yy} &= \sum_{i=1}^N \frac{y_i^2}{\sigma_i^2}
 \end{aligned}$$

qui, à  $S$  près, sont des moyennes, variances et covariances.

$$\text{alors } \begin{cases} aS + bS_x &= S_y \\ aS_x + bS_{xx} &= S_{xy} \end{cases} \quad \text{d'où } \begin{cases} a = \frac{S_{xx}S_y - S_x S_{xy}}{\Delta} \\ b = \frac{S S_{xy} - S_x S_y}{\Delta} \end{cases} \quad (10)$$

où  $\Delta = S S_{xx} - (S_x)^2$  est le déterminant du système et  $\Delta/S^2$  est la « variance » des  $x_i$  (déterministes!).

# Qualité de la régression : variances/covariances de $a$ et $b$

**Résidus**  $r_i =$  déviations à l'optimum :

$$r_i = y_i - f(x_i, \vec{p}^a)$$

$\sum_{i=1}^N r_i^2 / \sigma_i^2 \rightarrow \chi^2$  à  $N - M = N - 2$  degrés de liberté

Variances et covariances de  $a$  et  $b$  (rappel :  $x_i$  déterministes) :

$$\text{Var}(a) = \frac{S_{xx}}{\Delta} \quad \text{Var}(b) = \frac{S}{\Delta} \quad \text{Cov}(a, b) = -\frac{S_x}{\Delta} . \quad (11)$$

**Coefficient de corrélation de  $a$  et  $b$  :**

$$R_{ab} = \frac{\text{Cov}(a, b)}{\sigma_a \sigma_b} = -\frac{S_x}{\sqrt{S S_{xx}}}$$

$|R_{ab}| \leq 1$  doit être aussi petit que possible

$R_{ab} = \pm 1$  annulent  $\Delta$  et font que  $a$  et  $b$  ne sont plus définis.

# Qualité de la régression : coefficient de corrélation linéaire

Réécriture du modèle :

$$x = a' + b' y$$

+ hypothèses simplificatrices sur  $\sigma_i$  :

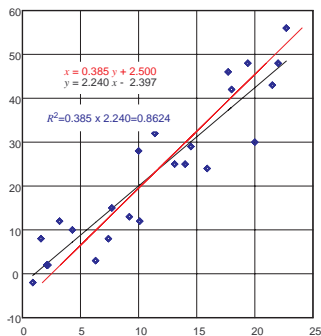
$$b' = \frac{S S_{xy} - S_x S_y}{S S_{yy} - S_y^2} .$$

Droites confondues  $\Leftrightarrow b \times b' = 1$ ,

*i.e.* le **coefficient de corrélation linéaire** :

$$r^2 = b \times b' = [\text{Cov}(x, y)]^2 / (\sigma_x^2 \sigma_y^2) = 1$$

La corrélation linéaire est donc appréciée **qualitativement** selon que  $r^2$  est plus ou moins proche de  $\pm 1$ .



# Cas linéaire général

Modèle  $y = f(x, \vec{p})$  linéaire **par rapport aux paramètres**  $\vec{p}$  :

$$f(x, \vec{p}) = \sum_{k=1}^M p_k F_k(x) , \quad (12)$$

où  $F_1(x) \cdots F_M(x)$  sont  $M$  fonctions quelconques de  $x$ .

La minimisation de  $\mathcal{E}(\vec{p}) = \sum_{i=1}^N \left( \frac{y_i - \sum_{k=1}^M p_k F_k(x_i)}{\sigma_i} \right)^2$  (13)

donne les  $M$  équations normales :

$$\sum_{i=1}^N \frac{y_i - \sum_{j=1}^M p_j F_j(x_i)}{\sigma_i} \times \frac{F_k(x_i)}{\sigma_i} = 0 \text{ pour } k = 1, \dots, M. \quad (14)$$

# Équations normales linéaires

En développant comme dans le cas 2-D, on obtient sous forme matricielle :

$$\sum_{j=1}^M A_{kj} p_j = b_k \quad \text{soit} \quad A \cdot p = b \quad (15)$$

où les coefficients  $A_{kj}$  de la matrice  $M \times M$  symétrique  $A$  et les composantes du vecteur-colonne  $b$  sont :

$$A_{kj} = \sum_{i=1}^N \frac{F_k(x_i) F_j(x_i)}{\sigma_i^2} \quad \text{et} \quad b_k = \sum_{i=1}^N \frac{y_i F_k(x_i)}{\sigma_i^2} \quad (16)$$

Le vecteur des paramètres  $p$  peut alors être obtenu avec l'une des méthodes standard de résolution des systèmes linéaires, sous la forme :

$$p = A^{-1} \cdot b$$



# Structure de la matrice A et du vecteur b

$A_{ij}$  et  $b_i$  peuvent se calculer à partir de  $G_{ik} = F_i(x_k)/\sigma_k$  et  $Y_k = y_k/\sigma_k$  :

$$\begin{cases} A_{ij} = \sum_{k=1,N} G_{ik} G_{jk} \\ b_i = \sum_{k=1,N} G_{ik} Y_k \end{cases} \quad \text{soit} \quad \begin{cases} A = G^t G \\ b = G Y \end{cases} \quad \text{donc} \quad p = (G^t G)^{-1} \cdot G \cdot Y$$

de façon plus visuelle :

$$A = \underbrace{\begin{bmatrix} G_{11} & \cdots & G_{1N} \\ \vdots & & \vdots \\ G_{M1} & \cdots & G_{MN} \end{bmatrix}}_{\text{taille } N \gg M} \underbrace{\begin{bmatrix} G_{11} \dots G_{M1} \\ \vdots \\ \vdots \\ G_{1N} \dots G_{MN} \end{bmatrix}}_{\text{taille } M \ll N} \quad \text{et} \quad b = \underbrace{\begin{bmatrix} G_{11} & \cdots & G_{1N} \\ \vdots & & \vdots \\ G_{M1} & \cdots & G_{MN} \end{bmatrix}}_{\text{taille } N \gg M} \underbrace{\begin{bmatrix} Y_1 \\ \vdots \\ \vdots \\ Y_N \end{bmatrix}}_{\text{taille } 1}$$

**NB :** La matrice A ( $M \times M$ ) est symétrique et définie positive (somme des carrés des  $\varepsilon_i$ ), ce qui assure l'existence d'un minimum unique.

# Variances-covariances : méthode probabiliste

Une fois obtenue la solution, il faut estimer sa qualité. Une première chose est de voir à quel point elle est **contrainte**, en évaluant la **co-variance des paramètres** : c'est la matrice  $C = A^{-1}$  qui les donne.

En effet, de  $p = C G Y \Rightarrow p_i = \sum_{m,k} C_{im} G_{mk} Y_k$  on tire :

$$\begin{aligned} \Rightarrow \text{Cov}(p_i, p_j) &= \sum_{m,n=1}^M \sum_{k,l=1}^N C_{im} G_{mk} C_{jn} G_{nl} \underbrace{\text{Cov}(Y_k, Y_l)}_{\delta_{kl}} \\ &= \sum_{m,n} C_{im} C_{jn} \underbrace{\sum_k G_{nk} G_{mk}}_{A_{mn}} = [C A C]_{ij} = C_{ij} , \end{aligned}$$

en utilisant successivement l'hypothèse d'indépendance des déviations  $\{\varepsilon_k\}$  et la symétrie de A et donc de C.

**Ce résultat s'appliquera de la même façon dans le cas non linéaire.**

# Variances-covariances : méthode algébrique

Expression de la vraisemblance  $\mathcal{L}$  autour du point  $\vec{p}^a$  :  
en notant  $\delta\vec{p} = \vec{p} - \vec{p}^a$ , on a :

$$\mathcal{L}(\delta\vec{p}) \propto \exp\left(-\frac{1}{2} {}^t\delta\mathbf{p} \cdot \mathbf{A} \cdot \delta\mathbf{p}\right).$$

Décroît d'autant plus vite que  $\mathbf{A}$  est grand (minimum très piqué).

# Variances-covariances : méthode algébrique

Expression de la vraisemblance  $\mathcal{L}$  autour du point  $\vec{p}^a$  :  
 en notant  $\delta\vec{p} = \vec{p} - \vec{p}^a$ , on a :

$$\mathcal{L}(\delta\vec{p}) \propto \exp\left(-\frac{1}{2} {}^t\delta\mathbf{p} \cdot \mathbf{A} \cdot \delta\mathbf{p}\right).$$

Décroît d'autant plus vite que  $\mathbf{A}$  est grand (minimum très piqué).

Diagonalisation de  $\mathbf{A}$  sous la forme  $\mathbf{A} = \mathbf{U} \cdot \mathbf{D} \cdot \mathbf{U}$  ( $\mathbf{U}$  orthogonale,  $\mathbf{D}$  diagonale)  $\rightarrow$  obtention des paramètres  $q_i$ , tels que  $\vec{q} = \mathbf{U}\delta\vec{p}$ , **indépendants** et de moyenne nulle. On a alors :

$$\mathcal{L} \propto \prod_{i=1}^M \exp\left(-\frac{1}{2} D_{ii} q_i^2\right) \quad \Rightarrow \quad \text{Var}(q_i) = 1/D_{ii} .$$

# Variances-covariances : méthode algébrique

Expression de la vraisemblance  $\mathcal{L}$  autour du point  $\vec{p}^a$  :  
 en notant  $\delta\vec{p} = \vec{p} - \vec{p}^a$ , on a :

$$\mathcal{L}(\delta\vec{p}) \propto \exp\left(-\frac{1}{2} {}^t\delta\mathbf{p} \cdot \mathbf{A} \cdot \delta\mathbf{p}\right).$$

Décroît d'autant plus vite que  $\mathbf{A}$  est grand (minimum très piqué).

Diagonalisation de  $\mathbf{A}$  sous la forme  $\mathbf{A} = \mathbf{U} \cdot \mathbf{D} \cdot \mathbf{U}$  ( $\mathbf{U}$  orthogonale,  $\mathbf{D}$  diagonale)  $\rightarrow$  obtention des paramètres  $q_i$ , tels que  $\vec{q} = \mathbf{U}\delta\vec{p}$ ,  
 indépendants et de moyenne nulle. On a alors :

$$\mathcal{L} \propto \prod_{i=1}^M \exp\left(-\frac{1}{2} D_{ii} q_i^2\right) \quad \Rightarrow \quad \text{Var}(q_i) = 1/D_{ii}.$$

En revenant aux paramètres  $p_i$ , et en utilisant l'indépendance des  $q_i$ , on obtient :

$$\langle \delta p_i \delta p_j \rangle = \left[ \mathbf{U} \cdot \mathbf{D}^{-1} \cdot \mathbf{U} \right]_{ij} = C_{ij}$$

qui est le résultat énoncé précédemment.

# Cas non-linéaire : approche générale

- Le cas général est non-linéaire : ajustement de la largeur  $w$  d'une lorentzienne ou d'une gaussienne, ou de la fréquence d'une sinusoïde
- Problème : dans les équations normales :

$$\sum_{i=1}^N \left( \frac{y_i - f(x_i, \vec{p}^a)}{\sigma_i^2} \right) \frac{\partial f(x_i, \vec{p}^a)}{\partial p_k} = 0,$$

les dérivées partielles  $\partial f(x_i, \vec{p}) / \partial p_k$  dépendent du point  $\vec{p}$  considéré dans l'espace des paramètres.

- Méthodes **itératives** : diminution progressive de l'erreur quadratique pondérée  $\mathcal{E}(\vec{p})$  jusqu'à obtenir un minimum **local**.

# Méthode itérative : principe

Valeur initiale  $\vec{p}^{(0)}$  ("guess")  $\longrightarrow \vec{p}^{(1)}, \dots, \vec{p}^{(n)}$ .

Arrêt lorsque  $\mathcal{E}(\vec{p}^{(n)})$  cesse de diminuer  $\rightarrow$  **Critère de convergence** :

$$0 \leq \mathcal{E}(\vec{p}^n) - \mathcal{E}(\vec{p}^{n+1}) \leq \eta_{\text{tol.}} \quad \text{où} \quad \eta_{\text{tol.}} \ll \mathcal{E}(\vec{p}^{(n)}) . \quad (17)$$

## Difficultés :

- Convergence vers **minimum local** dénué de sens  $\rightarrow$  choix de  $\vec{p}^{(0)}$  crucial et méthode adaptée si plusieurs minima locaux possibles.
- $\mathcal{E}(\vec{p})$  peut présenter des vallées dans lesquelles on descend très vite, et où ensuite la progression est **très lente** (méandres).
- Nombre très élevé d'évaluations de la fonction  $f$  et de ses dérivées, très coûteuses en temps de calcul si  $N$  est important.

Cela est d'autant plus probable que le nombre  $M$  de paramètres est élevé.  
 $\rightarrow$  procéder par étapes en faisant varier certains paramètres seulement.

# La méthode du gradient

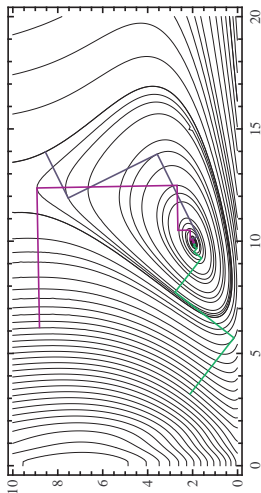
Méthode la plus simple : déplacement  $\delta \vec{p} = \vec{p}^{n+1} - \vec{p}^n$  dans la direction opposée au gradient de  $\mathcal{E}$ , qui définit la **plus grande pente** (“steepest descent”) de  $\mathcal{E}(\vec{p})$  :

$$\delta \vec{p} = -\gamma \nabla \mathcal{E}(\vec{p}^n) \quad (\gamma \in \mathbb{R}^+) ,$$

Le gradient est défini par ses composantes  $\partial \mathcal{E} / \partial p_k$  (termes à gauche dans les équations normales, à un facteur  $-2$  près).

Méthode assez **robuste** si le pas est assez petit, mais dans ce cas elle converge **très lentement**.

**Problème** : comment choisir  $\gamma$  ?



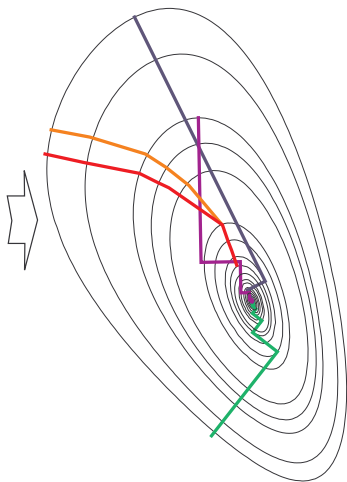
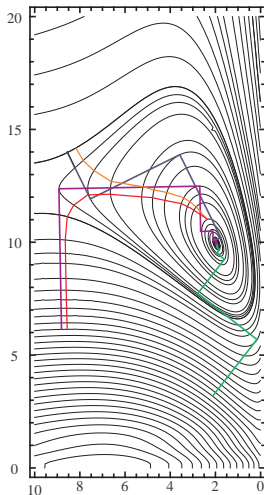


# Gradient : le choix du pas

Ni la minimisation en ligne (direction fixe tant qu'on descend donc cheminement avec angles droits)

ni la méthode à petits pas (changements de direction faibles)

ne sont efficaces.



# La matrice hessienne

Gain de temps : considération de la **courbure locale** pour choisir la **direction** de  $\delta \vec{p}$ , via la matrice **hessienne** :

$$\begin{aligned}
 [\mathbf{H}(\vec{p})]_{kl} &= \frac{\partial^2 \mathcal{E}(\vec{p})}{\partial p_k \partial p_l} \\
 &= 2 \sum_{i=N}^M \frac{1}{\sigma_i^2} \left( \frac{\partial f(x_i, \vec{p})}{\partial p_k} \frac{\partial f(x_i, \vec{p})}{\partial p_l} - (y_i - f(x_i, \vec{p})) \frac{\partial^2 f(x_i, \vec{p})}{\partial p_k \partial p_l} \right)
 \end{aligned}$$

C'est son **anisotropie** qui est le principal responsable de la lenteur de convergence de la méthode du gradient.

# Prise en compte de la courbure

En faisant un développement limité de  $\mathcal{E}(\vec{p})$  autour de  $\vec{p}^{(n)}$  :

$$\mathcal{E}(\vec{p}) \approx \mathcal{E}(\vec{p}^{(n)}) + \nabla \mathcal{E}(\vec{p}^{(n)}) \cdot \delta \vec{p} + \frac{1}{2} {}^t \delta \vec{p} \cdot \mathbf{H} \cdot \delta \vec{p} + \dots, \quad (18)$$

En dérivant, le gradient au point  $\vec{p}^{(n+1)} = \vec{p}^{(n)} + \delta \vec{p}$  s'écrit :

$$\nabla \mathcal{E}(\vec{p}^{(n+1)}) = \nabla \mathcal{E}(\vec{p}^{(n)}) + \mathbf{H} \cdot \delta \vec{p}$$

Pour qu'il s'annule, il faut choisir  $\delta \vec{p}$  tel que :

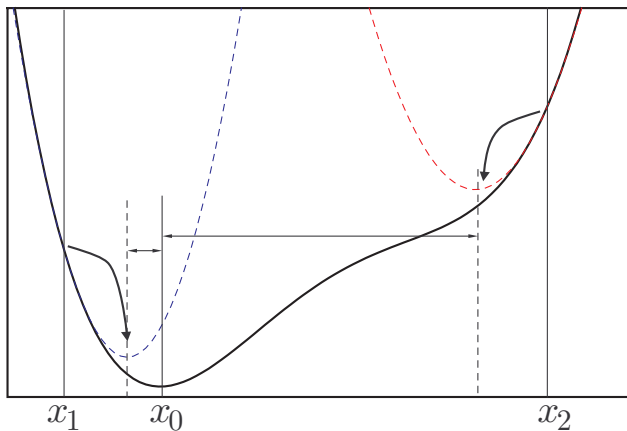
$$\nabla \mathcal{E}(\vec{p}^{(n)}) + \mathbf{H} \cdot \delta \vec{p} = 0 \quad \Leftrightarrow \quad \delta \vec{p} = -\mathbf{H}^{-1} \cdot \nabla \mathcal{E}(\vec{p}^{(n)}). \quad (19)$$

Si  $\mathcal{E}$  **quadratique**  $\rightarrow$  solution directe.

(cas linéaire :  $\mathbf{H} = 2\mathbf{A}$ ,  $\nabla \mathcal{E} = -\mathbf{b}$ , indépendantes du point  $\vec{p}$  considéré).

Cette solution est bien plus rapide lorsqu'on est assez proche d'un minimum, (*i.e.* au fond de la vallée).

## Illustration à une dimension



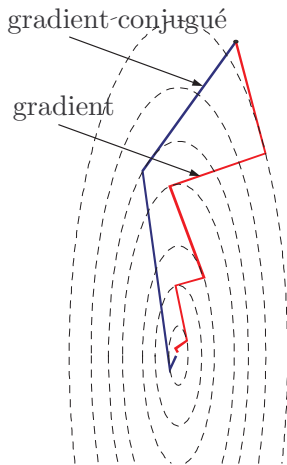
# La méthode du gradient conjugué

Elle consiste à faire des « pas » ou incréments  $\Delta \vec{p}$  dans la direction du  $\delta \vec{p}$  tel que défini par l'équation (19).

Elle est ainsi nommée car la surface de niveau et l'incrément sont « conjugués », *i.e.* ne sont pas orthogonaux pour le produit scalaire normal, mais pour la forme quadratique  $H$ , soit :

$$\vec{u} \cdot \nabla \mathcal{E} = 0 \quad \Rightarrow \quad {}^t \delta \vec{p} H \vec{u} = 0 .$$

Elle a le mérite de progresser dans la « bonne » direction, et conduit à une convergence rapide si l'approximation quadratique (18) est adaptée.



*Comparaison des méthodes du gradient et du gradient conjugué*

# Taille des pas et choix de la méthode

Choix de la taille des pas  $\leftarrow$  courbure :

Par ex. :  $\gamma_i \approx \alpha/H_{ii}$ ,  $\alpha < 1$  .

N.B. : Facteur  $\gamma_i$  différent dans chaque direction.

Dans ces conditions :

- La **méthode du gradient** est plus **sûre** quand on est loin de l'optimum car l'approximation quadratique est alors grossière.
- La **méthode du gradient conjugué** est plus **rapide** lorsqu'on s'approche du minimum, car l'approximation quadratique est alors bien meilleure.

**NB** : Combinaison des deux méthodes (descente dans la vallée puis progression rapide)  $\implies$  **Levenberg-Marquardt** avec un paramètre permettant de passer continûment d'une méthode à l'autre.

# Méthode du gradient conjugué : les équations

- Si les déviations  $\epsilon_k = y_k - f(x_k, \vec{p})$  sont petites, indépendantes et dispersées selon une loi normale centrée, la contribution à H des termes croisés en  $\frac{\partial^2 f}{\partial p_k \partial p_l}$ ,  $k \neq l$  est négligeable.
- On **définit** alors la matrice A et le vecteur B comme suit :

$$A = G^t G, \quad b = G E \quad \text{où} \quad G_{ik} = \frac{1}{\sigma_k} \frac{\partial f(x_k, \vec{p})}{\partial p_i} \quad \text{et} \quad E_k = \frac{\epsilon_k}{\sigma_k} \quad (20)$$

- Parallélisme avec la méthode dans le cas linéaire. A et b dépendent du point considéré mais **tendent assez vite vers leur valeur au minimum.**

# Méthode du gradient conjugué : l'algorithme

Les itérations sont alors effectuées comme suit :

- ① **Choisir** une valeur *initiale*  $\vec{p}^{(0)}$  et évaluer  $\mathcal{E}(\vec{p}^{(0)})$ .
- ② Au point  $\vec{p}^{(n)}$ , **évaluer** A, b et **résoudre** l'équation  $A \delta p = b$ .
- ③ Prendre  $\vec{p}^{(n+1)} = \vec{p}^{(n)} + \delta \vec{p}$ .
- ④ **Tester** la convergence : si  $\mathcal{E}$  **ne diminue plus**, c'est à dire si

$$0 < \mathcal{E}(\vec{p}^{(n)}) - \mathcal{E}(\vec{p}^{(n+1)}) < \eta \ll 1,$$

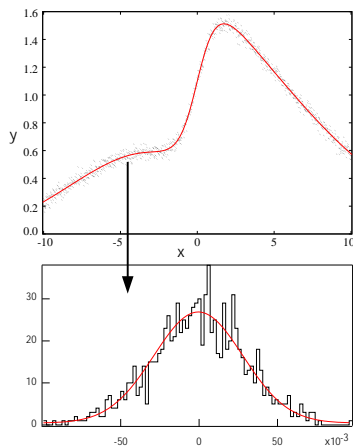
on peut **arrêter** les itérations. Sinon, **retourner** en (2).

**Variante** : Méthode plus robuste avec  $\delta \vec{p} \leftarrow \gamma \delta \vec{p}$ ,  $0 < \gamma < 1$  et contrôle de  $\gamma$  en fonction de l'évolution de  $\mathcal{E}$  à chaque itération.



# Critères d'évaluation de la qualité de l'ajustement

- Matrice **variances-covariances** :  $A^{-1}$ .  
 Évalue la **contrainte** de l'ajustement.  
 Variances souvent très faibles et non-significatives.  
 Attention aux covariances  $\equiv$  choix des paramètres.
- **Dépendance en  $x$**  et **distribution statistique** des résidus  $\rightarrow$  vraisemblance du modèle et qualité de l'ajustement.



La **pertinence du modèle** est enfin appréciée par le **test** du  $\chi^2$  (erreur évaluée au minimum !), avec  $N - M$  degrés de liberté, dont la fonction de répartition est proportionnelle à la fonction  $\Gamma$  incomplète.

# Test du $\chi^2$

On calcule alors :

$$Q\left(\frac{N-M}{2}, \frac{\chi^2}{2}\right) = 1 - \frac{\Gamma\left(\frac{N-M}{2}, \frac{\chi^2}{2}\right)}{\Gamma\left(\frac{N-M}{2}\right)},$$

Mesure la probabilité que des résidus ( $\propto$  loi normale) **donnent une valeur supérieure ou égale** à la valeur  $\chi^2$  obtenue.

Seuils à utiliser  $\longleftrightarrow$  **risque** toléré.

Typiquement :  $Q \geq 0.1 \rightarrow$  confiance au modèle,  $Q \sim 0.01$  valeur juste passable.

Note : valeur trop grande de  $\chi^2$  ( $\leftrightarrow$  trop faible de  $Q$ )  $\Leftarrow$  **dispersions  $\sigma_i$  sous-évaluées** (trop optimiste sur les barres d'erreur).