Généralités

Normes et rayon spectral:

Définition norme vectorielle :

$x \mapsto \|x\|$, $R^n \to R_+$ vérifiant :

- $\|x\| \geq 0$ et $\|x\| = 0 \Leftrightarrow x = 0$
- $\|\alpha.x\| = |\alpha|.\|x\|$
- $\|x + y\| \leq \|x\| + \|y\|$

Exemple de norme vectorielle :

- Norme 1 : $\|X\|_1 = \sum_i |x_i|$
- Norme 2 : $\|X\|_2 = \sqrt{\sum_i |x_i|^2} = \sqrt{\langle X, X \rangle}$
- Norme ∞ : $\|X\|_\infty = \underset{i}{Sup} |x_i|$

Proposition : En dimension finie, toutes ces normes sont équivalentes ; en pratique, on choisit celle qui nous arrange.

Définition norme matricielle :

C'est une norme dans l'espace vectorielle $M_n(R)$, avec en plus $\|A \times B\| \leq \|A\|.\|B\|$.

Norme matricielle induite :

A partir d'une norme vectorielle, on construit une norme matricielle induite :

$$\|\|A\|\| = \underset{X \neq 0}{Sup}\left(\frac{\|AX\|}{\|X\|}\right) = \underset{\|X\|=1 \text{ ou } \|X\| \leq 1}{Sup} \left(\|AX\|\right).$$

Notons de plus que $\|\|I\|\| = 1$.

En particulier pour la norme 2, on a la relation : $\|\|I\|\|_2 = \sqrt{\rho\left(A^t A\right)}$ avec ρ le rayon spectral.

Propriétés :

- $\|AX\| \leq \|\|A\|\|.\|X\|$ et $\|\|AB\|\| \leq \|\|A\|\|.\|\|B\|\|$
- Cas de A symétrique : $\|\|A\|\|_2 = \rho(A) = \max_i |\lambda_i|$
- Dans le cas général, $\|\|A\|\|_2 = \mu_{\max}$, maximum des valeurs singulières $\mu_i = \sqrt{\lambda_i}$ avec λ_i les valeurs propres (positives) de la matrice symétrique $A^t A$. (?)
- Soit A une matrice carré quelconque n x n. Pour toutes normes de matrice induite, on a $\rho(A) \leq \|\|A\|\|$.

Conditionnement d'une matrice inversible :

Considérons un système linéaire AX=b. L'expérience de Wilson met un évidence une instabilité des calculs dans certains cas. Si l'on modifie sensiblement les paramètres de la matrice A, ou du second membre b, on peut trouver des résultats complètement différends !

- $Cond(A) = \|\|A\|\|.\|\|A^{-1}\|\|$

- $Cond(\alpha A) = Cond(A)$
- $Cond(A) \geq 1$ avec $Cond(Id) = 1$
- Soit une matrice Q orthogonale : $\|QX\|_2 = \|X\|_2$ et $Cond(Q) = 1$
- Les bons conditionnements sont les petits conditionnements (au mieux 1 pour les matrices orthogonales).
- Pour A symétrique : $Cond_2(A) = \frac{|\lambda|_{\max}}{|\lambda|_{\min}}$; dans la cas général : $Cond_2(A) = \frac{|\mu|_{\max}}{|\mu|_{\min}}$

Théorèmes sur le conditionnement :

Théorème : $AX = B$ et $A(X + \Delta X) = b + \Delta b$. On a $\frac{\|\Delta X\|}{\|X\|} \leq Cond(A)\frac{\|\Delta b\|}{\|b\|}$.

Théorème : $AX = B$ et $(A + \Delta A)(X + \Delta X) = b$. On a $\frac{\|\Delta X\|}{\|X + \Delta X\|} \leq Cond(A)\frac{\|\Delta A\|}{\|A\|}$.

Remarque : Le conditionnement traduit l'amplification des grandeurs relatives.

Inverse numérique d'une matrice :

Soit A une matrice inversible et A^{-1} son inverse théorique.
M est un inverse de A du point de vue numérique si :

- $\frac{\|M - A^{-1}\|}{\|A^{-1}\|}$ est petit, ce qui correspond à $M=A^{-1}$
- $\frac{\|M^{-1} - A\|}{\|A\|}$ est petit, ce qui correspond à $A=M^{-1}$
- $\|AM - Id\|$ et $\|MA - Id\|$ sont petits, ce qui correspond à $AM - Id = 0$ et $MA - Id = 0$

On s'adapte au calcul que l'on veut faire. Notons que l'on résout rarement un système linéaire en calculant l'inverse $X = A^{-1}b$

Systèmes linéaires Ax=b : méthodes directes

Formules de Crammer

Cette méthode nécessite le calcul d'un déterminant, dont la complexité est en *n!*. La compléxité de la formule de Cramer est en $n^2 n!$. Par conséquent, cette méthode n'est pas utilisable en informatique.

Méthode du pivot de Gauss

À la k^{ème} étape : pour $i > k$, $\text{Ligne i} - \left(\frac{a_{i,k}}{a_{k,k}} \right) \text{Ligne k}$. On pose $l_{i,k} = \frac{a_{i,k}}{a_{k,k}}$.

L'algorithme à une complexité en $\frac{n^3}{3}$.

Le problème des coefficients diagonaux nuls : il faut permuter les lignes lorsque apparaît un zéro sur la diagonale.

Stratégie du pivot maximum partiel :

On permute la k^{ème} ligne et la i^{ème} ligne avec i tel que $a_{i,k} = \underset{l \geq k}{Sup} \left| a_{l,k} \right|$. Pour des raisons de stabilité numérique, on divise par le terme de la colonne k, qui a la plus grande valeur absolue.

Stratégie du pivot avec seuil :

On effectue une permutation uniquement des $\left| a_{k,k} \right| < \varepsilon$. Cependant cette méthode peut être mise en défaut. Il vaut mieux utiliser la méthode du pivot maximum partiel, qui n'est pas vraiment plus coûteuse en calcul.

Calcul de l'inverse

Une idée pour résoudre le sytème Ax=b serait de calculer l'inverse de A. En effet, on a $x = A^{-1}b$.

Méthode de Shermann – Morison

Soit A une matrice n x n, réelle inversible, dont on connaît l'inverse A^{-1} ; soit B une perturbation, on cherche à calculer $(A+B)^{-1}$.

On démontre que si $B = XY^t$, alors $(A+B)^{-1} = (I + c.A^{-1}B)A^{-1} = A^{-1} + c.A^{-1}BA^{-1}$, avec $c = \frac{-1}{1+Y^t A^{-1} X}$.

On suppose maintenant que la perturbation ne modifie qu'un seul en (i, j) : $B = \varepsilon.E_{i,j} = \underbrace{\varepsilon.e_i}_{X} \times \underbrace{e_j}_{Y}^{t}$.

Notons $A^{-1} = (\alpha_{k,j})_{1 \leq k,l \leq n}$. Alors on trouve $\left[(A + \varepsilon.E_{i,j})^{-1} \right]_{k,l} = \alpha_{k,l} - \varepsilon \frac{\alpha_{k,i}\alpha_{j,l}}{1+\varepsilon.\alpha_{j,i}}$.

Elimination de Gauss

On applique la stratégie du pivot partiel maximum, au calcul de l'inverse.
On considère une matrice A inversible, de dimension n

On résout $Ax = e_l$ pour $1 \leq l \leq n$, ce qui revient à écrire $A.A^{-1} = I$. On procède par élimination de Gauss, ce qui donne une matrice triangulaire supérieure (complexité en n^3). Ensuite, on réalise n remontées, une par vecteur e_l (complexité en n^2).

En résumé, on a $Ax = e_l$ *et* $U.x = M.e_l$ avec $U = MA$ *une matrice triangulaire supérieure.*

On veut traiter tous les seconds membres en une seule passe ; pour cela on considère la matrice n x 2n suivante : $\left(\begin{array}{c|c} A & I \end{array}\right)$ qui après calculs donne $\left(\begin{array}{c|c} I & A^{-1} \end{array}\right)$.

On donne ici un exemple d'écriture en langage de description algorithmique :

```
Procédure pivot ( k , l : entiers ; OK : booléens )
  Début
  Max := |A(k,k)| ;
  l := k ;
  Pour i := k+1 à n  faire
   Si max < |A(i,k)|
  Alors
     Début
     Max := |A(i,k)| ;
     l := i ;
     Fin ;
  OK := Max > ε ;
  Fin ;
```

```
Procédure permuter ( k , l : entiers )
  Début
  Si k ≠ l
  Alors
     Pour j := k à 2n  faire
        Début
        c := A(k, j) ;
        A(k, j) : = A(l, j) ;
        A(l, j)  := c ;
       Fin ;
Fin ;
```

On donne maintenant le programme principal, réalisant le calcul de l'inverse.

```
Début
Initialiser A et lire ε ;

/* triangularisation */
Pour k := 1 à n  faire
  Début
  Pivot (k , l , OK ) ;
  Si non(OK)
  Alors
     Début
     Ecrire « matrice non inversible » ;
     Exit echec ;
     Fin ;
  Permuter ( k , l ) ;
  Pour j := k+1 à 2n  faire
     A(k, j):= A(k, j)/ A(k,k) ;
  Pour i := k+1 à n  faire
     Pour j := k+1 à 2n  faire
```

$A(i,j) := A(i,k) - A(i,k) * A(k,j)$;
Fin ;

/ n remontées */*
Pour k := 1 à n faire
Pour i := n-1 à 1 par pas de -1 faire
Début
s := 0 ;
Pour j := i+1 à n faire
$s := s + A(i,j) * A(j,n+k)$;
$A(i,n+k) := A(i,n+k) - s$;
Fin ;
Fin ;

Remarques :
- Choix du pivot : pour des raisons de stabilité numérique, on divise toujours par le terme de la colonne k qui a la plus grande valeur absolue (*Cf. Pivot partiel maximum*).
- Recherche du pivot : si la plus grande valeur absolue est inférieure à la précision machine ε, alors on arrête en disant que *la matrice n'est pas inversible à ε près* (test d'inversibilité informatique). Cependant, elle peut très bien être inversible du point de vue mathématique !

Factorisation A=LU

Soit A une matrice régulière, de dimension n.

Lien avec la méthode de Gauss

On applique la méthode de Gauss sans pivoter. U est une matrice triangulaire supérieure réelle, telle que $\prod_{k=1,n} J_k A = U$.

On donne la définition des matrices $J_k = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & \alpha_{k+1,k} & 1 & \\ & \vdots & & \ddots \\ & \alpha_{n,k} & & 1 \end{pmatrix}$ avec $\alpha_{l,k} = -\frac{a_{l,k}}{a_{k,k}}$ pour $l > k$.

On démontre que l'inverse des J_k existe et que $L = \prod_{k=1,n} J_k^{-1}$ est une matrice triangulaire inférieure, avec des 1 sur la diagonale. En fait, on a $J_k^{-1} = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & -\alpha_{k+1,k} & 1 & \\ & \vdots & & \ddots \\ & -\alpha_{n,k} & & 1 \end{pmatrix}$.

Calcul algorithmique des coefficients de L et de U

On procède par identification en utilisant les propriétés de L (triangulaire inférieure avec une diagonale de 1) et de U (triangulaire supérieure). On cherche *un algorithme par ligne* donnant les $l_{i,j}$ pour $1 \le j \le i-1$ et les $u_{i,j}$ pour $1 \le j \le n$.

Pour i := 1 à n faire
Début
Pour j := 1 à i-1 faire

$$l_{i,j} := \left(a_{i,j} - \sum_{k=1}^{j-1} l_{i,k}.u_{k,j} \right) \Big/ a_{j,j} \ ;$$

/ avec* $l_{i,i} = 1$ **/*

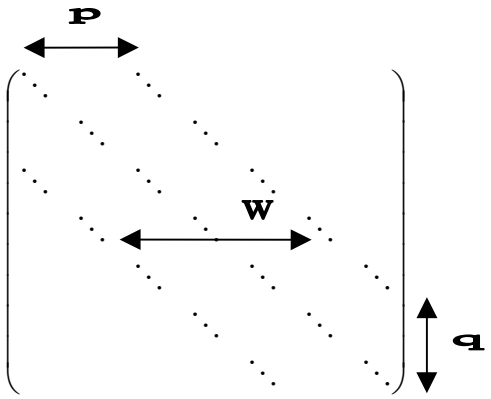
Pour j := i à n faire

$$u_{i,j} := a_{i,j} - \sum_{k=1}^{i-1} l_{i,k}.u_{k,j} \ ;$$

Fin ;

Remarque : L et U écrasent la matrice A.

Cas particulier des matrices à « *structure bande* »



- La largeur de la bande est W = P+Q+1
- On ne stocke dans une structure de donnée appropriée que la bande, soit un O(W.N), ce qui est intéressant si W << N.

Propriété fondamentale :
La bande se conserve par factorisation LU. On parle de *structure de données statique*. En revanche, si on utilise des techniques de pivotage au cours d'une factorisation LU sur une matrice bande, alors la structure bande n'est pas conservée.
On utilise la méthode uniquement dans le cas, où à priori on ne pivote pas. La condition suffisante la plus connue est « A matrice à diagonale strictement dominante », c'est à dire $\forall i \in 1,n \quad |a_{i,i}| > \sum_{\substack{j=1,n \\ j \neq i}} |a_{i,j}|$.

Cas des matrices symétriques définies positives (SDP)

Cf. factorisation Cholesky et Crout.

Factorisation PA=LU

P est la matrice des permutations.
L est une matrice triangulaire inférieure dont la diagonale se compose de 1.
U est une matrice triangulaire supérieure.

Matrice de permutation élémentaire i $\leftrightarrow$ j :

$$P_{i,j} = \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & & 0 & & 1 & \\ & & & & 1 & & \\ & & & & & \ddots & \\ & & & & & & 1 \\ & & & 1 & & 0 & \\ & & & & & & 1 \\ & & & & & & & \ddots \\ & & & & & & & & 1 \end{pmatrix} \begin{matrix} \\ \\ \\ \rightarrow \text{ligne i} \\ \\ \\ \\ \rightarrow \text{ligne j} \\ \\ \\ \end{matrix}$$

Le produit PA permute les lignes i et j de la matrice A. Le produit AP permute les colonnes i et j de la matrice A.

(à compléter avec le cours : obtention de la factorisation et utilisation de la factorisation)

Factorisation $A=BB^t$: Cholesky

Cette factorisation s'applique pour des matrices A symétriques et positives.
Théorème : Soit A une matrice SDP de dimension n. Il existe une et une seule matrice B triangulaire inférieure, telle que $A=BB^t$.

Factorisation $A=LDL^t$: Crout

Théorème : Soit A une matrice SDP de dimension n. Il existe une et une seule matrice L triangulaire inférieure et D diagonale positive, telles que $A=LDL^t$.
Remarque : Crout a le même comportement numérique que Cholesky. Cependant, il est plus utilisé car généralisable au cas complexe.

Soit A une matrice symétrique définie positive (quelconque ?).
L est une matrice triangulaire inférieure de dimension n dont la diagonale se compose de 1.
D est une matrice diagonale de dimension n.

L est en fait la matrice B pour laquelle les termes d'une colonne sont divisés par les coefficients b_{ii}. D est la matrice dont les coefficients diagonaux sont les b_{ii}^2.

Algorithme par ligne pour l'obtention de la factorisation

On écrit $L=\begin{pmatrix} 1 & 0 & 0 \\ & \ddots & 0 \\ l_{i,j} & & 1 \end{pmatrix}$ et $D=\begin{pmatrix} d_1 & & \\ & \ddots & \\ & & d_n \end{pmatrix}$.

Il s'agit de calculer les $l_{i,j}$ et les d_i. On a $a_{i,j}=\sum_{k=0}^{j-1} l_{ik} d_k l_{jk} + l_{ij} d_j$.

On établit un algorithme par ligne : Pour la 1ère ligne, on écrit $d_1 = a_{1,1}$. Supposons maintenant le calcul effectué jusqu'à la ligne i-1 ; le calcul de la ligne i est donné par $l_{i,j}=a_{i,j}-\frac{\sum_{k=1}^{j-1}\left(l_{ik} d_k l_{jk}\right)}{d_j}$ pour $j \le i-1$ et par $d_i = a_{i,i} - \sum_{k=1}^{j-1}\left(l_{ik}^2 d_k\right)$. Ainsi une CNS pour cette factorisation est que les d_i ne soient pas nuls.

On donne l'algorithme correspondant :

```
Pour i de 1 à n faire
  Début
  Pour j de 1 à i-1 faire
     l_{i,j} := a_{i,j} - Σ_{k=1}^{j-1} (l_{ik} d_k l_{jk}) / d_j ;
   /* l_{i,i} := 1 */
   d_i := a_{i,i} - Σ_{k=1}^{j-1} (l_{ik}^2 d_k) ;
  Fin ;
```

Remarque : L et D écrase la partie triangulaire inférieure de A.

On veut maintenant apporter une amélioration à cet algorithme en diminuant les nombres de calcul ; on économise les produits $l_{i,k}.d_k$, que l'on stocke pour un i fixé dans le vecteur C[k].

On donne l'algorithme modifié :

Pour i de 1 à n faire
Début
Pour j de 1 à i-1 faire
Début

$C[j] := a_{i,j} - \sum_{k=1}^{j-1} l_{jk}.C[k]$;

$l_{i,j} := \frac{C[j]}{d_j}$;

Fin ;

$d_i := a_{i,i} - \sum_{k=1}^{j-1} l_{ik}.C[k]$;

Fin ;

Algorithme par colonne pour l'obtention de la factorisation

Principe : On construit L et D, en procédant par décomposition successives. On réalise le découpage suivant :

$A = A_1 = \left(\begin{array}{c|c} d_1 & v_1^t / d_1 \\ \hline v_1 / d_1 & B_1 \end{array}\right)$, avec v_1 un vecteur colonne de dimension n-1, et B_1 une matrice carré de dim n-1.

On pose $L_1 = \left(\begin{array}{c|c} 1 & O \\ \hline v_1 / d_1 & I_{n-1} \end{array}\right)$, $A_2 = \left(\begin{array}{c|c} d_1 & O \\ \hline O & B_1' \end{array}\right)$, avec $B_1' = B_1 - \frac{v_1.v_1^t}{d_1}$.

Ainsi, on a : $A = A_1 = L_1 A_2 L_1^t$. Puis, en réitérant n-1 fois, il vient : $A = A_1 = \underbrace{(L_1 \cdots L_{n-1})}_{L} \underbrace{A_n}_{D} \underbrace{(L_1 \cdots L_{n-1})^t}_{L^t}$.

Le produit des matrices $L_k = \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & & \ddots & \\ & & v_k / d_k & & 1 \end{pmatrix}$ donnent la matrice finale

$L = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & \ddots & \\ v_1 / d_1 & v_2 / d_2 & \cdots & 1 \end{pmatrix}$.

Ce qui correspond en effet à une factorisation de Crout, car la matrice $D = (d_i)$ est bien diagonale et la matrice L est bien triangulaire inférieure, par construction.

On donne l'algorithme de calcul de L et de D par colonne, qui procède en écrasant la partie triangulaire inférieure de A, (les 1 de la diagonale de L ne sont pas stockés, car on préfère y stocker D) :

Début
Pour k de 1 à n-1 faire
  Pour j de k+1 à n faire
    Début
    $d := a_{j,k} / a_{k,k}$;
    Pour i de j à n faire $a_{i,j} := a_{i,j} - a_{i,k}.d$;
    $a_{j,k} := d$;
  Fin ;
Fin ;

La structure donnée d'implémentation serait un vecteur à une dimension stockant consécutivement les colonnes de la partie triangulaire inférieure de A.

Remarque fondamentale : $a_{i,j}$ est modifié par $a_{i,k}$ et $a_{j,k}$.

Etude du remplissage lors de la factorisation de matrices SDP creuses

Considérons une matrice A symétrique creuse. On utilise une structure de donnée type profil, il est important de savoir où se trouvent les termes à priori non nuls de L.
Une matrice est dite *creuse* si elle contient « un grand nombre » de termes nuls. Plus précisément, on considère que la matrice est creuse à partir du moment, où son pourcentage de termes nuls permet un gain informatique par rapport à un traitement classique qui la considérerait pleine.

Définitions :

- *Un terme de remplissage* apparaît en *(i,j)* au cours de la factorisation si et seulement si $a_{i,j} = 0$ et $l_{i,j} \neq 0$. On reformule le problème comme la conservation du creux initial de la matrice.
- $l_{i,j}$ est *logiquement non nul* si $a_{i,j} \neq 0$ ou $\exists k, 1 \leq k \leq j-1$ tel que $l_{i,k} l_{j,k} \neq 0$. C'est à dire, à la k^{éme} itération $(1 \leq k \leq n-1)$ un terme $a_{i,j}$ de la matrice A qui est nul, devient non nul ssi $a_{j,k} \neq 0$ et $a_{i,k} \neq 0$ avec $k \leq j-1$. *Preuve* (algorithme par colonne): $\underbrace{a_{i,j}}_{\neq 0} := \underbrace{a_{i,j}}_{=0} - \frac{\overbrace{a_{i,k}.a_{j,k}}^{\neq 0}}{a_{k,k}}$.
- Les $l_{i,j}$ logiquement non nuls se trouvent à l'intérieur du profil de A.

Exemple de remplissage total : $E_1 = \begin{pmatrix} x & & & & \\ x & x & & & \\ x & 0 & x & & \\ x & 0 & 0 & x & \\ x & 0 & 0 & 0 & x \end{pmatrix} \rightarrow E_2 = \begin{pmatrix} x & & & & \\ x & x & & & \\ x & x & x & & \\ x & x & x & x & \\ x & x & x & x & x \end{pmatrix}$

Le coefficient (3,2) dépend uniquement des coefficients (3,1) et (2,1) qui sont non nuls, par conséquent le coefficient (3,2) devient logiquement non nul : *effet de remplissage*. Ensuite, le coefficient (4,2) qui dépend des coefficients (4,1) et (2,1) devient logiquement non nul. (…) Le coefficient (5,3) dépend des coefficients (5,k) et (3,k) pour k variant de 1 à 3-1=2, etc.

Remarque : Ainsi, on voit que le phénomène dépend de l'ordre selon lequel on exécute les itérations, c'est à dire de la numérotation des inconnues du système ! Si l'on permute les indices 5 et 1 dans l'exemple précédent, on

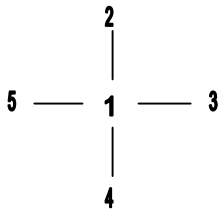
obtient la matrice suivante : $E_3 = \begin{pmatrix} x & & & & \\ 0 & x & & & \\ 0 & 0 & x & & \\ 0 & 0 & 0 & x & \\ x & x & x & x & x \end{pmatrix}$ qui ne produit pas de remplissage !

La question qui se pose maintenant est de savoir si l'on peut trouver une matrice de permutation P, telle que la factorisation de la matrice PAP^t crée le moins de remplissage possible ? Il n'existe pas de méthode exacte en temps raisonnable et on va utiliser des méthodes approchées (heuristiques).

Caractérisation du remplissage par les graphes

On utilise un *graphe* afin de déterminer quels sont les termes logiquement non nuls qui vont apparaître au cours du calcul.
Le graphe associé à une matrice A symétrique d'ordre n possède n sommets que sont les indices de 1 à n des coefficients de la matrice. Un coefficient non nul est repéré dans le graphe par un arc entre les indices de ligne et de colonne de ce coefficient.

Exemple : Graphe associé à la matrice E_1 et à la permutation identité :

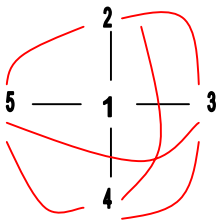


Considérons une permutation donnée telle que $\sigma(x_i) = i$ et plaçons nous à la k^éme^ itération :

- trouver le monotone adjacent de x_k qui est l'ensemble de tous les sommets liés à x_k avec un numéro plus grand, $Madj(x_k) = \{x_j, j > k \text{ et tel que } (x_j, x_k) \text{soit une arête possible}\}$
- rajouter *une arête de remplissage* pour tout couple de tels sommets non liés entre eux.
- On fait cela dans l'ordre des k croissants de 1 à n-1. A la fin, on obtient *le graphe dit d'élimination.*

Exemple :

- $Madj(x_1) = \{x_2, x_3, x_4, x_5\}$ → remplissage total
- $Madj(x_2) = \{\ \}$ → pas de remplissage
- $Madj(x_3) = \{\ \}$ → pas de remplissage
- $Madj(x_4) = \{\ \}$ → pas de remplissage



Profil ligne

Considérons une matrice SDP A de dimension n.

Définition : Pour chaque ligne, on considère la colonne c_i correspondant au premier terme non nul sur la ligne. La contribution de la ligne i au *profil* est $P_i = \{a_{i,j}, c_i \le j \le i\}$ qui sera effectivement stockée par ordre croissant de colonne (soit $i - c_i + 1$ coefficients). Le profil ligne est l'union des P_i pour i variant de 1 à n. On a besoin d'un vecteur de dimension n donnant pour tout i la valeur de c_i.

La *taille du profil* est $m = \sum_{i=1}^{n} i - c_i + 1$.

Propriété fondamentale : Le profil ligne se conserve par factorisation (*statique*). Le remplissage est inclus dans le profil ligne. Par conséquent, on va chercher à trouver des numérotations qui tendent à minimiser la taille du profil ligne.

On adapte l'algorithme de factorisation : $l_{i,j} := a_{i,j} - \sum_{k=\max(c_i,c_j)}^{j-1} \left(l_{ik} d_k l_{jk}\right) / d_j$, ainsi on diminue le volume de calcul.

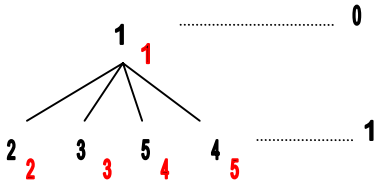
Optimisation du profil ligne

Critère approché : On va chercher une numérotation des sommets du graphe G_σ minimisant le profil A. Pour cela, on cherche à diminuer la taille *m*. On utilise *un critère local* : à i fixé, on va essayer d'obtenir c_i proche autant que possible de i, ce qui va induire *un tassement vers la diagonale.*

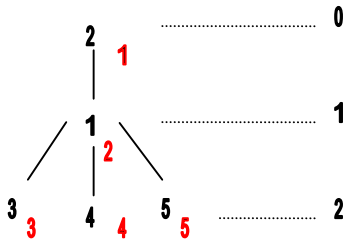
Le problème en terme de numérotation sur G_σ est le suivant : « numéroter ses voisins avec des numéros les plus proches possibles de i ».

Numérotation en couches :
On partitionne l'ensemble des sommets du graphe par rapport à leur distance à un sommet de départ *x*. On numérote continûment selon les couches et le plus grand écart varie comme un O(longueur d'une couche). Le nombre de sommets étant fixé, on cherche le sommet $\bar{x}$ de départ de telle manière que le nombre de couche $h(\bar{x})$ générés à partir de $\bar{x}$ soit maximum ; ceci aura tendance à minimiser la longueur des couches.

Exemple : prenons *x* = 1, on a *h(x) = 1*.



Algorithme :
- Choisir x quelconque
- Construire la structure en couche à partir de *x*
- Choisir un sommet *y* de la couche terminale ayant le moins de voisins et construire les couches à partir de ce nouveau sommet, nécessairement $h(y) \le h(x)$
- Si *h(x) > h(y)* Alors réitérer, Sinon, on est dans le cas où *h(x) = h(y)*, et *y* est le point de départ.



A partir de l'exemple précédent, on choisit *y = 2* et on obtient *h(y) = 2*. En réitérant de nouveau, on obtiendra pas mieux. Donc on déduit $\bar{x} = 2$.

On applique le changement de numérotation :

Ancien numéro i	1	2	3	4	5
Nouveau numéro $\sigma(i)$	2	1	3	4	5

Ce qui donne la transformation suivante : $E_1 = \begin{pmatrix} x & & & & \\ x & x & & & \\ x & 0 & x & & \\ x & 0 & 0 & x & \\ x & 0 & 0 & 0 & x \end{pmatrix} \rightarrow E_4 = \begin{pmatrix} x & & & & \\ x & x & & & \\ 0 & x & x & & \\ 0 & x & 0 & x & \\ 0 & x & 0 & 0 & x \end{pmatrix}$.

Limitation du remplissage

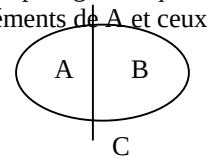
Théorème : CNS de création d'une arête de remplissage

Soit G_σ le graphe numéroté associé à la matrice A.

Une arête de remplissage (x_i, x_j) est créée si et seulement si :

- l'arête est déjà présente dans G_σ ;
- il existe un chemin allant de x_i à x_j tel que les numéros intermédiaires soient plus petits que ceux des extrémités.

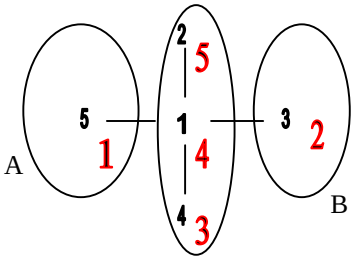
Idée de numérotation : Pour mettre en défaut ce théorème, on utilise *des séparateurs topologiques*. Cette méthode consiste à séparer deux sous-ensembles A et B par un ensemble C, appelé *séparateur ;* de sorte que tout *chemin* allant d'un élément de A vers un élément de B passe au moins par un sommet de C. Si on numérote les sommets de C avec des numéros plus grands que ceux affectés à A et B, alors il n'y aura pas d'arête de remplissage créées entre les éléments de A et ceux de B.



On applique ce principe de numérotation *récursivement par bloc*, jusqu'à tomber sur une grille qui n'est plus séparable. Ce faisant on limite excellemment le remplissage, mais le profil risque d'être mauvais.

Remarque : Il faut une autre structure de donnée que celle du profil non adapté. Par ailleurs, il y a aussi une indépendance des calculs exploitable sur une machine à architecture parallèle.

Exemple :
On reprend l'exemple précédent…



On effectue le changement de numérotation : $E_1 = \begin{pmatrix} x & & & & \\ x & x & & & \\ x & 0 & x & & \\ x & 0 & 0 & x & \\ x & 0 & 0 & 0 & x \end{pmatrix} \rightarrow E_5 = \begin{pmatrix} x & & & & \\ 0 & x & & & \\ 0 & 0 & x & & \\ 0 & 0 & 0 & x & \\ x & x & x & x & x \end{pmatrix}$ et l'on peut vérifier que cette dernière matrice ne se remplie pas du tout !

Utilisation de la factorisation pour résoudre Ax=b

Il s'agit de résoudre $Ax = b$ en utilisant la factorisation de Cholesky - CROUT : $ALDL^t X = b$. On procède en trois étapes.

- *Descente* : Posons $y = DL^t x$ et résolvons $Ly = b$, dont la complexité est en $\frac{n^2}{2}$ (matrice triangulaire).

  On écrit un algorithme utilisant un accès par ligne dans la matrice L.

  $y_1 := b_1$;

  Pour i de 2 à n faire $y_i = b_i - \sum_{j=1}^{i-1} l_{i,j}\, y_j$;

- *Résolution du système diagonal* : Posons $z = L^t x$ et résolvons $Dz = y$ (par n divisions).

Pour i de 1 à n faire $z_i = \frac{y_i}{di}$;

- *Montée :* Résolvons finalement $L^t x = z$, dont la complexité est en $\frac{n^2}{2}$.

  On écrit un algorithme utilisant un accès par ligne dans L^t, ce qui revient à un accès par colonne dans L.

  $x_n := y_n$;

  Pour i de n-1 à 1 faire $x_i := z_i - \sum_{j=i+1}^{n} {l_{i,j}}^t x_j$;

On veut exploiter la symétrie. On ne stocke que L et D ; et on ne stocke pas les 1 de la diagonale de L, qui sont pris en compte directement dans le calcul.

Considérons le fait que l'on stocke les données en mémoire de façon mono-dimensionnelle. On va donc être obligé de choisir une organisation ligne par ligne, ou colonne par colonne des données.
Si l'on choisit une organisation ligne par ligne, on s'aperçoit que l'algorithme de descente est bien adapté à la structure de données par ligne, tandis que l'algorithme de montée, qui pratique un accès aux données par colonne ne l'est pas. C'est pourquoi, il faut écrire un algorithme de montée accédant aux données par ligne et non plus par colonne.

Pour i de 1 à n faire $x_i := z_i$;
Pour i de n à 2 par pas de -1 faire
  Pour j de i-1 à 1 par pas de -1 faire
    $x_j := x_j - l_{i,j} x_i$;

Systèmes linéaires Ax=b : méthodes itératives

Principe

Méthode itératives classiques :

On cherche à obtenir une suite de vecteur $(X_n)_n$ convergeante vers $\overline{X}$ telle que $A\overline{X} = b$.

En posant $A = M - N$ où M est une matrice inversible, on propose la formule de récurrence $MX_{n+1} = NX_n + b$ ou encore $X_{n+1} = M^{-1}NX_n + M^{-1}b$. Si $\overline{X}$ est solution du problème, il vient que $(\overline{X} - X_n) = (M^{-1}N)^n \times (\overline{X} - X_0)$. $B = (M^{-1}N)$ est appelée *la matrice d'itération*.

Ainsi pour que $X_n \to \overline{X}$, il est nécessaire que $(M^{-1}N)^n \to 0$.

Remarque : Le cadre normale de convergence des méthodes itératives classiques est : « A matrice symétrique définie positive ».

Vitesse de convergence :

Théorème : $(B^n)_n \to 0$ ssi $\rho(B) < 1$.

Vitesse de convergence : Considérons *la matrice d'itération* B symétrique, alors $\|\|B\|\|_2 = \rho(B)$ et l'on obtient le résultat suivant : $\|x_k - x\|_2 \leq \rho(B)^k . \|x_0 - x\|_2$.

En conclusion, le rayon spectral de la matrice d'itération mesure la vitesse de convergence. La convergence est d'autant plus rapide que le rayon spectral est plus petit. On admettra que ce résultat ce généralise aux matrices quelconques.

Méthode Jacobi

Notation : $A = \begin{pmatrix} & & -F \\ & D & \\ -E & & \end{pmatrix}$

On pose $A = M - N$ avec M = D et N = E + F selon le schéma ci-dessus. Ce qui donne la formule de récurrence suivante : $X_{n+1} = D^{-1}(E+F)X_n + D^{-1}b$.

Soit $x_k = (x_k^i)_{i=1,n}$ le vecteur itéré, on donne l'algorithme donnant x_{k+1} en fonction de x_k.

Pour i de 1 à n faire

$$x_{k+1}^i = \frac{-1}{a_{i,i}} \left[\sum_{j=1}^{i-1} a_{i,j} x_k^j + \sum_{j=i+1}^{n} a_{i,j} x_k^j - b_i \right]$$

Théorème : Soit A une matrice symétrique, à diagonale strictement dominante, c'est à dire $a_{i,i} > \sum_{i \neq j} |a_{i,j}|$ alors la méthode est convergente car $\rho(D^{-1}(E+F)) < 1$.

A priori, cette méthode converge pour une matrice A SDP.

Méthode Gauss-Seidel

Boucle directe

On reprend le schéma de matrice précédent et on pose $M = D - E$ et $N = F$. La formule de récurrence s'écrit : $(D-E)X_{n+1} = FX_n + b$.

On calcule x_{k+1}^i, l'i[ème] composante du vecteur X_{k+1} : $a_{i,i}x_{k+1}^i = -\sum_{j=1}^{i-1} a_{i,j}x_{k+1}^j - \sum_{j=i+1}^{n} a_{i,j}x_k^j + b_i$.

On propose l'algorithme suivant :

Pour i de 1 à n faire

$$x_{k+1}^i = \frac{1}{a_{i,i}}\left[-\sum_{j=1}^{i-1} a_{i,j}x_{k+1}^j - \sum_{j=i+1}^{n} a_{i,j}x_k^j + b_i\right]$$

On procède par écrasement du vecteur X_k : $\left(\underbrace{x_{k+1}^j,\ 1 \le j \le i-1}_{\text{déjà calculés}} \;\middle|\; \underbrace{x_{k+1}^i}_{\text{calcul en cours}} \;\middle|\; x_k^j,\ i+1 \le j \le n\right)^t$.

Boucle rétrograde

Cette fois, on pose $M = D - F$ et $N = E$.

On propose l'algorithme suivant :

Pour i de n à 1 faire

$$x_{k+1}^i = \frac{1}{a_{i,i}}\left[-\sum_{j=1}^{i-1} a_{i,j}x_k^j - \sum_{j=i+1}^{n} a_{i,j}x_{k+1}^j + b_i\right]$$

Conclusion

Théorème : Soit A une matrice symétrique définie positive, alors la méthode est convergente car $\rho\left((D-E)^{-1}F\right) < 1$.

Comparaison : Si A est SDP les méthodes de *Jacobi* et de *Gauss-Seidel* convergent, et *Gauss-Seidel* converge plus vite que *Jacobi* car $\rho(J) = \rho(L)^2$ avec $J = D^{-1}(E+F)$ et $L_1 = (D-E)^{-1}F$ ou $(D-F)^{-1}E$.

Méthode de relaxation

Pour cette méthode, on pose $M = \frac{D}{w} - E$ et $N = M - A = \left(\frac{1-w}{w}\right)D + F$, avec $w \neq 0$. On donne la matrice d'itération : $L_w = \left(\frac{D}{w} - E\right)^{-1}\left[\left(\frac{1-w}{w}\right)D + F\right]$. Ainsi pour $w = 1$, on se ramène à la méthode de *Gauss-Seidel*.

Théorème : Soit A une matrice SDP. Il y a convergence si et seulement si $\rho(L_w) < 1$. On démontre que ceci est vrai si et seulement si $w \in]0,2[$.

Remarque :

- $w = 1$: *Gauss-Seidel*
- $w < 1$: *Sous-relaxation*
- $w > 1$: *Sur-relaxation*

Boucle directe

On veut calculer x_{k+1}^i : $\frac{a_{i,i}}{w} x_{k+1}^i = -\sum_{j=1}^{i-1} a_{i,j} x_{k+1}^j - \sum_{j=i+1}^{n} a_{i,j} x_k^j + \left(\frac{1}{w} - 1\right) a_{i,i} x_k^i + b_i$. Comme dans *Gauss-Seidel*, on note que l'algorithme procède par écrasement.

Pour i de 1 à n faire

$$x_{k+1}^i = (1-w) x_k^i - \frac{w}{a_{i,i}} \left[\sum_{j=1}^{i-1} a_{i,j} x_{k+1}^j + \sum_{j=i+1}^{n} a_{i,j} x_k^j \right] + \frac{w}{a_{i,i}} b_i$$

Choix de w :

Soit A une matrice symétrique définie positive, et tridiagonale par blocs : $A = \begin{pmatrix} & & O \\ & \iddots & \\ O & & \end{pmatrix}$.

On cherche un *w* optimal, c'est à dire qui rend $\rho\left(M^{-1}N\right)$ minimum. On établit dans le cas des matrices tridiagonales : $w_{optimal} = \frac{2}{1+\sqrt{1-\rho(L_1)^2}}$ et $\rho\left(L_{w_{optimal}}\right) = w_{optimal} - 1$.

Note : On utilise aussi cette méthode de façon heuristique pour des matrices qui ne sont pas tridiagonale.

Dichotomie : Si l'on cherche $w_{optimal}$ par dichotomie, l'expérience montre qu'il faut obligatoirement procédé par valeurs supérieures (la fonction $w \mapsto \rho(L_w)$ possède pour $w > w_{optimal}$ une pente de 1).

Évaluation du rayon spectral de L :

On écrit les itérés de *Gauss-Seidel* : $\begin{cases} x_{k+1} = L_1.x_k + c \\ x_k = L_1.x_{k-1} + c \end{cases}$ d'où $x_{k+1} - x_k = L_1.(x_k - x_{k-1})$. On utilise *la méthode de la puissance itérée* et on déduit : $\rho(L_1) = \lim_{k\to\infty} \frac{\|x_{k+1} - x_k\|}{\|x_k - x_{k-1}\|}$. En pratique, on fait l'approximation pour des itérés dont l'écart a diminué de manière significative.

Schéma d'implémentation :

- Soit x_0 donné
- On réalise p itérations avec *Gauss-Seidel*, on obtient la suite des itérées jusqu'à x_p à partir de laquelle on évalue de manière approchée $\rho(L_1)$ et $w_{optimal}$.
- Puis on applique la méthode de relaxation avec $w = w_{optimal}$.
- On s'arrête lorsque l'on a atteint la précision désirée.

Remarque : Toute la difficulté réside dans la choix de p. Il faut que le nombre total d'itérations soit $<< n$ pour être compétitif avec *Cholesky*. Le choix de p se fait au cas par cas.

Méthodes du gradient[1]

Principe : méthode de la plus grande descente

Soit A une matrice symétrique définie positive.

Théorème : Soit $J(X) = \langle AX, X \rangle - 2\langle b, X \rangle$. Alors $\overline{X}$ est solution de $AX = b$, si et seulement si $\overline{X}$ réalise le minimum de J.

En dimension 1, $A = a$ et $J(x) = ax^2 - bx$ (équation de parabole) ; d'où le minimum de J est $\frac{b}{a}$, ce qui correspond bien à la solution de l'équation linéaire $ax = b$.

Formule de récurrence : Prenons un X_0 quelconque, et supposons que nous ayons obtenu par itération X_k. Considérons la courbe de niveau k $\{X \text{ tel que } J(X) = J(X_k)\}$, pour calculer X_{k+1}, on se déplace du point X_k, appartenant là a courbe de niveau k, dans la direction de son gradient : $\overrightarrow{g_k} = \frac{1}{2}\overrightarrow{grad} = AX_k - b$. On donne ainsi la formule de récurrence : $X_{k+1} = X_k - \theta_k \overrightarrow{g_k}$. Il reste maintenant à choisir la valeur de θ_k, de sorte que J soit le plus petit possible. On montre que $J(X_{k+1}) = \theta_k{}^2 \langle Ag_k, g_k \rangle - 2\theta_k \langle g_k, g_k \rangle + J(X_k)$, donc il faut prendre $\theta_k = \frac{\langle g_k, g_k \rangle}{\langle Ag_k, g_k \rangle}$.

On résume le passage de X_k à X_{k+1} :

- $\overrightarrow{g_k} = AX_k - b$
- $\theta_k = \frac{\langle g_k, g_k \rangle}{\langle Ag_k, g_k \rangle}$
- $X_{k+1} = X_k - \theta_k \overrightarrow{g_k}$

Algorithme du gradient conjugué

Dans cette méthode, on va utiliser un ensemble de directions conjugués. $\overrightarrow{g_k}$ est la meilleure direction locale, mais pas à long terme, sinon l'algorithme convergerait d'un coup! D'où l'idée, d'apporter une correction pour le choix de la direction.

On pose $X_{k+1} = X_k + \alpha_k \overrightarrow{d_k}$ avec $\overrightarrow{d_k} = -\overrightarrow{g_k} + \beta_{k-1}\overrightarrow{d_{k-1}}$. En résolvant le système obtenu, par annulation des dérivées partielles pour la fonction J (qui doit être minimum), on calcule les valeurs des coefficients α_k et β_{k-1}.

On va maintenant résumer la méthode :

- *Initialisation* : X_0 quelconque, $g_0 = AX_0 - b$, et $d_0 = -g_0$.
- *Passage de l'indice k à l'indice k+1* : On suppose que l'on connaît X_k, d_k, et g_k. On calcule $\alpha_k = \frac{\langle g_k, g_k \rangle}{\langle Ad_k, d_k \rangle}$, d'où $X_{k+1} = X_k + \alpha_k d_k$.

[1] Cette partie a été rédigée en privilégiant les notes de cours plutôt que celle de TD, où les notations et les méthodes diffèrent sur quelques points.

- *On prépare la suite en calculant* : $g_{k+1} = g_k + \alpha_k A d_k$ ou $AX_{k+1} - b$; $\beta_k = \frac{\langle g_{k+1}, g_{k+1} \rangle}{\langle g_k, g_k \rangle}$ et $d_{k+1} = -g_{k+1} + \beta_k d_k$.

Remarque : On démontre que la correction apportée dans cette méthode est optimale.

Formule par récurrence : On démontre après coup que cette méthode donne la solution de l'équation linéaire au bout de N itérations ! Par conséquent, on pourrait la considérer comme une méthode directe, mais on préfère la prendre comme une méthode itérative, et s'arrêter pour $k \leq N$ quand la précision est satisfaisante.

Préconditionnement :

Convergence de la formule :

On établit que : $\left\| X_k - \overline{X} \right\|_A \leq \left(\frac{Cond(A)-1}{Cond(A)+1} \right)^k \left\| X_0 - \overline{X} \right\|_A$. Notons que plus *Cond(A)* est petit, plus la convergence est rapide. En particulier, si $Cond(A) = 1$, alors on converge d'un coup, c'est à dire X_0 est déjà solution ! Rappelons : $Cond(A) \geq 1$. Ici, on voit que la vitesse de convergence va dépendre de la matrice A, et plus particulièrement de son conditionnement. On va alors introduire la notion de *préconditionnement*.

Transformation :

On cherche à résoudre un autre système $\tilde{A}Y = \tilde{b}$ ayant même solution, mais pour lequel la matrice $\tilde{A}$ possède un meilleur conditionnement.

On considère $C = E.E^t$. On pose $\tilde{A} = E^{-1}.A.E^{-t}$ et $\tilde{b} = E^{-1}b$. Ainsi $AX = b \Leftrightarrow \tilde{A}Y = \tilde{b}$ pour $Y = E^t X$.

Remaque : Les algorithmes passant par l'intermédiaire de Y sont dits *non transformés*. Lorsque Y n'apparaît plus dans le calcul l'algorithme est dit *transformé*.

Algorithme du gradient préconditionné transformé :

Prenons un X_0 quelconque, et supposons que nous ayons obtenu par itération X_k. On pose $g_k = AX_k - b$ et $h_k = C^{-1} g_k$, ce qui revient à résoudre $Ch_k = g_k$. Résolution d'un système linéaire ! Certes. Mais simple, puisqu'on a supposé auparavant, que l'on avait une factorisation de Cholesky de la matrice C, c'est à dire $C = E.E^t$ (algorithme de complexité n²). Ensuite, on calcule $\tilde{\theta}_k = \frac{\langle g_k, h_k \rangle}{\langle Ah_k, h_k \rangle}$. Et on passe de X_k à X_{k+1} en faisant $X_{k+1} = X_k - \tilde{\theta}_k h_k$.

Ainsi, on obtient le résultat fondamental suivant $\left\| X_k - \overline{X} \right\|_A \leq \left(\frac{Cond(\tilde{A})-1}{Cond(\tilde{A})+1} \right)^k \left\| X_0 - \overline{X} \right\|_A$. Il faut à présent travailler pour que le conditionnement de Ã soit meilleur que celui de A. Or, nous avons posé $\tilde{A} = E^{-1} A E^t$, par où nous pouvons déduire que $Cond(\tilde{A}) = Cond(C^{-1}A)$, car ces deux matrices ont les mêmes valeurs propres.

Choix de C (méthode heuristique) :

- *Factorisation incomplète de Cholesky* : Lorsque l'on pose $A = BB^t$, il peut sur venir un phénomène de remplissage ! D'où l'idée de réaliser une factorisation incomplète $A = EE^t + R$. On ne calcule dans la matrice E que les $l_{i,j}$ tels que $a_{i,j} \neq 0$, pour lesquels donc on évite le phénomène de remplissage.

  Si l'on pose $C = BB^t = A$, on obtient un conditionnement de 1 (le meilleur) car $C^{-1}A = Id$; cependant il peut se produire un phénomène de remplissage peut souhaitable. Et si l'on pose $C = Id$, le conditionnement n'est pas amélioré par rapport à celui de *A*.

On va alors prendre $C = EE^t$ (*factorisation incomplète de Cholesky*), ce qui correspond à un compromis entre le choix d'un bon conditionnement pour Ã et la disparition du phénomène de remplissage.

- *Factorisation incomplète de Crout*: On peut tout aussi bien utiliser *Crout* (qui est la méthode proposée en TD). Dans ce cas, $A = LDL^t - R$ et on prend $C = LDL^t$.
  Voici deux conditions suffisantes chaques à l'existence : A est une matrice SDP à diagonale dominante, ou A est une matrice SDP avec $a_{i,j} \leq 0$ pour $i \neq j$.

Algorithme du gradient conjugué préconditionné transformé

- *Factorisation incomplète de Cholesky* : $A = EE^t + R$; on pose $C = EE^t$
- *Au départ* : On dispose d'un X_0 quelconque, de $g_0 = AX_0 - b$, de $h_0 = C^{-1}g_0$ et de $d_0 = -h_0$.
- *Passage de l'indice k à l'indice k+1* : On suppose que l'on connaît X_k, d_k, g_k et h_k.
  On calcule : $\alpha_k = \frac{\langle g_k, g_k \rangle}{\langle Ad_k, d_k \rangle}$ et $X_{k+1} = X_k + \alpha_k d_k$.
- *On prépare la suite en calculant* :
  - $g_{k+1} = g_k + \alpha_k Ad_k$ ou $= AX_{k+1} - b$
  - $h_{k+1} = C^{-1}g_{k+1}$ (simple à résoudre car factorisation de *Cholesky* ou de *Crout*)
  - $d_{k+1} = -h_{k+1} + \beta_k d_k$

Tests d'arrêts pour les méthodes itératives

On se donne une précision $\varepsilon > 0$. On voudrait que l'algorithme se termine une fois la précision atteinte, c'est à dire $\left\| X_n - \overline{X} \right\| \leq \varepsilon$. Cependant, on ne connaît pas $\overline{X}$! On utilise par conséquent le test suivant $\left\| X_{n+1} - X_n \right\| \leq 2\varepsilon$, dont on considère en pratique qu'il est équivalent au premier, même si cela n'est qu'à demi vrai en théorie (la réciproque est fausse). Il peut y avoir de rare cas où l'algorithme s'arrête, alors que l'on est loin de la solution !!! Mais faute de mieux…
Finalement, on prend comme test d'arrêt : $\left\| X_{n+1} - X_n \right\| \leq \varepsilon$ et /ou $\left\| AX_n - X_n \right\| = \left\| g_n \right\| \leq \varepsilon$.

Remarque : La méthode du gradient conjugué préconditionné est plus performant en nombre d'itération que la méthode de relaxation avec $w_{optimal}$.

Problèmes au moindres carrés (linéaire)

Régression linéaire

Etant donné une série de *M* points (x_i, y_i), on cherche une relation de la forme $y_i \cong ax_i + b$. On cherche *a* et *b* qui minimise l'erreur $\|erreur\|^2 = \sum_M {e_i}^2$ avec $e_i = y_i - (ax_i + b)$.

Cas général

On étudie un cas plus général $f(t_i) \cong \sum_N x_j . \phi_j(t_i)$, avec *N* inconnues x_j. On a *M* équations avec $M >> N$. On pose $A = (\phi_j(t_i))_{M \times N}$ et $b = (f(t_i))_M$. L'erreur est $Ax - b$. On cherche les x_j qui minimise cette erreur $\|Ax - b\|_2^2 = \sum_{i=1\ldots M} \left(\sum_{j=1\ldots N} a_{ij} x_j - b_i \right)^2$.

Méthode des équations normales

Théorème

Le vecteur $\overline{x}$ réalise le minimum de $\|Ax - b\|_2$ si et seulement si $\overline{x}$ est solution de $(A^t A)x = A^t b$ (équation normale). Il existe toujours au moins une solution. La solution est unique si et seulement si le rang de A est *N*.

Défauts numériques

On appliquera la méthode des équations normales pour un nombre faible d'inconnues 2, 3, 4 ou 5. Un premier défaut est l'imprécision du calcul engendré par le produit matriciel $A^t A$. Par ailleurs, si la matrice A est creuse, $A^t A$ peut très bien ne pas l'être…

Méthode de la factorisation QR (rectangulaire)

Définition (*M > N*)

$A_{M \times N} = Q_{M \times M} R_{N \times N}$, avec *Q* une matrice orthogonale et $R = \begin{pmatrix} \overline{R}_{N \times N} \\ \hline O \end{pmatrix}$, avec $\overline{R}$ une matrice triangulaire supérieure.

Théorème : minimiser $\|Ax - b\|_2$ pour *N* supérieur à 3 ou 4

Soit $Q^t b = \begin{pmatrix} c_1 \\ \hline c_2 \end{pmatrix}_M$ avec c_1 de dimension *N*. Le vecteur $\overline{x}$ minimise $\|Ax - b\|_2$ si et seulement si $\overline{x}$ est solution de $\overline{R}x = c_1$.

Obtention de la factorisation *A = QR* par *Householder*

Définition des matrices de *Householder*

Soit *u* un vecteur unitaire $\left(\|u\|_2 = 1\right)$ de dimension *m*. On définit la matrice de *Householder* de dimensio $m \times m$ $H_u = Id_m - 2u.u^t$. H_u est symétrique et orthogonale $\left(\|H_u\|_2 = 1\right)$. On note également $H_u = Id_m - \frac{v.v^t}{\beta}$ en posant $v = \sqrt{2\beta}.u$.

Propriété fondamentale des matrices de *Householder*

Soit *a* un vecteur de dimension *m*, non nul. Il existe *H*, une matrice de *Householder*, telle que $H \times a = \begin{pmatrix} \alpha \\ \hline O \end{pmatrix}_m$ avec α un réel.

Preuve :

- $\alpha^2 = \|a\|_2^2$
- On choisit le signe de α : le signe opposé à a_1.
- On pose $\beta = \alpha^2 - \alpha.a_1 > 0$.
- On choisit $v = \begin{pmatrix} \frac{a_1 - \alpha}{a_i} \end{pmatrix}_m$.

Principe de l'algorithme de factorisation

On veut obtenir la matrice $R_{M \times N}$ à partir de la matrice $A = A_0$. On procède colonne par colonne.

On rappelle la géométrie de $R = \left(\begin{array}{c|ccc} \alpha_1 & x & & \\ \hline 0 & \alpha_2 & & \\ & & \ddots & \\ \vdots & & & \\ & & & \\ 0 & & & O \end{array}\right)$.

- Occupons nous de la première colonne de A_0. Il existe une matrice de *Householder* H_1 de dimension *m* telle que $H_1 \times A_0 = A_1$, où la première colonne de A_1 est $\left(\begin{array}{c} \alpha_1 \\ \hline 0 \\ \vdots \\ 0 \end{array}\right)$.

- On s'intéresse maintenant à la 2ème colonne. Il existe une matrice de *Householder* $\tilde{H}_2$ de dimension *m-1* telle que $H_2 \times A_1 = A_2$, avec $H_2 = \begin{pmatrix} 1 & 0 \\ 0 & \tilde{H}_2 \end{pmatrix}$. Ainsi la première colonne de A_2 reste inchangée par rapport à A_1 et l'on a fabriqué la deuxième colonne $\begin{pmatrix} x \\ \alpha_2 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$.
- Il vient $A = A_0 = H_1 \times A_1 = H_1 \times H_2 \times A_2 = \ldots$ On itère la méthode *N* fois et il l'on obtient la factorisation $A = (H_1 \cdots H_n) A_n$ avec $Q = H_1 \cdots H_n$ et $R = A_n$.

Obtention de $Q^t b = c_1$ au cours de l'algorithme de factorisation

En vérité, on peut habilement intégrer le calcul de $Q^t b$ à l'algorithme précédent. Cela est simple, il suffit de rajouter une *N+1* colonne à la matrice initiale, $A_0 = (A \quad b)$. Ainsi après *N* itérations, on obtiendra le résultat $Q^t b$ toujours dans la même colonne.

Obtention de la factorisation *A = QR* par *Givens*

(…)

Résolution des équations non linéaires

Méthode de dichotomie

Considérons une fonction *f* en dimension 1. On isole une racine unique dans un intervalle $[a,b]$. On divise l'intervalle en deux à chaque étape, on identifie le sous - intervalle contenant la racine en comparant le signe du milieu $f(m)$avec $f(a)$. Et on réitère. La précision obtenue au bout de *k* itérations est $\frac{b-a}{2^k}$. La méthode de la dichotomie est robuste, mais n'est pas généralisable à la dimension *n*.

Généralités

Méthode convergente, d'ordre p

Une méthode itérative est convergente : $X_n \to \overline{X}$ solution, si $\exists C, p / \left\| X_n - \overline{X} \right\| \le C \left\| X_{n-1} - \overline{X} \right\|^p$. On dit que la méthode est dite d'ordre *p*.

Théorème du point fixe

Soit ϕ *k*-lipschitzienne contractante : $\left\| \phi(y_2) - \phi(y_1) \right\| \le k \left\| y_2 - y_1 \right\|$ avec $k < 1$. Soit X_0 et $X_{n+1} = \phi(X_n)$. Alors on a $X_n \to \overline{X}$ tel que $\overline{X} = \phi(\overline{X})$.

Pour appliquer ce théorème, il suffit de vérifier qu'il existe une boule $B\left(\overline{X}, \left\| X_0 - \overline{X} \right\|\right)$ sur laquelle, on a la propriété $\left\| \phi(y_2) - \phi(y_1) \right\| \le k \left\| y_2 - y_1 \right\|$ avec $0 < k < 1$. L'inégalité des accroissement finis nous donne $k = \sup_c \left| \phi'(c) \right| = \sup_{c,i,j} \left| \frac{\partial \phi_i(c)}{\partial x_j} \right|$.

Méthode d'itérations successives pour résoudre $F(X)=0$

Posons $\phi(X)=X-F(X)$. Si $\overline{X}$ est point fixe de ϕ alors $\overline{X}$ est solution de $F(X)=0$. On définit alors la récurrence suivante : $X_{n+1}=X_n-F(X_n)$.

Méthode de *Newton* - tangente

En dimension 1 pour des réels, $x_{n+1}=x_n-\frac{f(x_n)}{f'(x_n)}$. La méthode de la tangente est d'ordre 2. Inconvénient, si on part d'une certaine boule , la méthode converge, sinon l'algorithme peut osciller et ne pas converger ! Cette méthode se généralise à la dimension *n*. Pour des complexes, on peut employer deux fois cette méthode.

Méthode de *Newton - Raphson*

En dimension *n*, $X_{n+1}=X_n-\Delta_n$ où $J(F(X_n))\times\Delta_n=F(X_n)$ avec $J(F(X_n))=\left(\frac{\partial F_i(X_n)}{\partial x_j}\right)$ (matrice jacobienne). $J(F(X_n))$ est inversible et permet le calcul de Δ_n à chaque itération. Si on part d'une certaine boule au voisinage de la solution, la méthode est convergente d'ordre 2. Si on fait un choix au hasard pour X_0, il faut prévoir un test d'arrêt au bout de 100 itérations par exemple pour éviter le cas d'oscillations ! La complexité globale est en n^3.

Variante si *m* grand

La complexité globale passe en n^2 après la première utilisation… Factorisation *PA=LU*…

Méthode de *Bairstow* dans le cas des polynômes

On cherche à résoudre $P(x)=0$, avec $P(x)=x^n+a_{n-1}x^{n-1}+\ldots+a_0$.

Méthode

On cherche $x^2+\beta x+\alpha$ qui divise *P*. On a $P(x)=(x^2+\beta x+\alpha)Q(x)+(r_1x+r_0)$ (division euclidienne). Pour déterminer les coefficients α et β, on cherche une racine de l'application $F:(\alpha,\beta)\mapsto(r_0,r_1)$.

On veut appliquer la méthode de *Newton - Raphston* en dimension 2. Il faut d'abord calculer la matrice jacobienne *J* : on établit des formules par récurrence assez complexes… Puis, on applique effectivement la méthode de *Newton - Raphston*, à partir de (α_0,β_0), qui donne $(\overline{\alpha},\overline{\beta})$ (sous réserve que l'on se trouve initialement dans une boule suffisamment proche de la solution pour que la méthode soit effectivement convergente).

Finalement, le polynôme $A_1=x^2+\beta x+\alpha$ fournit deux racines de *P*. On réitère la méthode sur Q_1 tel que $P=A_1Q_1$. Cependant, les itérations successives entraînent une perte progressive de la précision.

Variante

Pour ne pas choisir (α_0,β_0) complètement au hasard, une méthode consiste à sélectionner une centaine de couples et à choisir, pour débuter l'algorithme, celui qui rend le polynôme minimum.

Amélioration de la précision

La racine obtenue étant assez proche de la racine réelle, on effectue la méthode de la tangente pour améliorer la précision. Tous les 4 ou 5 coups, on améliore la précision avec la précision avec la méthode de la tangente. Cette dernière méthode ne s'applique qu'aux racines réelles (on applique deux fois la méthode pour les racines complexes).

Méthode de Bernouilli pour le calcul de la racine dominante si elle est unique

???

Valeurs propres et vecteurs propres

Conditionnement

Théorème

Si *A* est diagonalisable en $D = P^{-1}AP$ avec *P* la matrice de passage. Pour tout δA, pour toute valeur propre λ de $A + \delta A$, il existe une valeur propre λ' de *A* telle que $|\lambda - \lambda| \leq Cond(P)\|\delta A\|$. Le meilleur cas possible est *A* symétrique, la matrice de passage est orthogonale et son conditionnement égale à 1.

Bissection de *Givens - Householder*

Méthode

1. Chercher une matrice tridiagonale semblable à *A* ayant même valeur propre.
2. Chercher ces valeurs propres.

Tridiagonalisation

Cet algorithme utilise des matrices de *Householder*. On procède bloc par bloc, en dimension décroissante.

- $A_1 = A$
- *1ère étape* : $A_2 = H_1 \times A_1 \times H_1^{-1}$ avec $H_1 = H_1^{-1} = \begin{pmatrix} 1 & O \\ O & \tilde{H}_1 \end{pmatrix}$ et $\tilde{H}_1$ la matrice de *Householder*. On découpe A_1 en bloc selon le schéma : $\left(\begin{array}{c|c} a_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right)$. Par la suite, $A_2 = \left(\begin{array}{c|c} a_{11} & \left(\tilde{H}_1 A_{12}\right)^t \\ \hline \tilde{H}_1 A_{21} & \tilde{H}_1 A_{22} \tilde{H}_1 \end{array}\right)$. On sait trouver $\tilde{H}_1$ tel que $\tilde{H}_1 A_{21} = \begin{pmatrix} \alpha_1 \\ \hline O \end{pmatrix}$ avec α un réel. Voilà ce qui en définitive va rendre la matrice tridiagonale.
- On applique la même méthode que précédemment avec une dimension en moins…
- On réitère la méthode *n* fois en totalité. Finalement, on obtient $A = (H_1 \dots H_n) \times A_n \times (H_1 \dots H_n)^{-1}$ où A_n est une matrice tridiagonale semblable à *A*.

Optimisation des calculs

On intègre le calcul de α dans la méthode et on optimise le produit des 3 matrices.

- $\alpha_1 = -Signe(a_{2,1}).\|A_{21}\|$
- $\beta = \alpha_1^{\,2} - \alpha_1 a_{2,1} > 0$
- $V = A_{21} - \begin{pmatrix} \alpha_1 \\ \hline O \end{pmatrix}$
- $W = \frac{1}{\beta} A_{22} V$
- $\lambda = \frac{1}{2\beta} V^t W$
- $Z = W - \lambda V$
- $VZ^t = \dots$ et $ZV^t = \dots$

- $\tilde{H}_1 A_{22} \tilde{H}_1 = A_{22} - \left(VZ^t + ZV^t\right)$

Méthode de recherche des valeurs propres

A partir de la matrice tridiagonale *M*, on va construire le polynôme caractéristique dont les racines sont précisément les valeurs propres.

Soit b_i les coefficients diagonaux, et c_i les coefficients sous-diagonaux. On a $P_0(\lambda)=1$, $P_1(\lambda)=b_1-\lambda$ et pour tout $i \le n$, $P_i(\lambda)=(b_i-\lambda)P_{i-1}(\lambda)-{c_{i-1}}^2 P_{i-2}(\lambda)$. Attention au cas particulier où un c_i est nul ! Il faut distinguer deux polynômes caractéristiques.

1. P_i est le polynôme caractéristique de M_i où M_i est la sous-matrice de dimension *i* extraite de *M* de telle sorte que $M_n = M$.
2. Le coefficient dominant de P_i est $(-1)^i \lambda^i$.
3. Si $P_i(\lambda_0)=0$ alors $P_{i+1}(\lambda_0)$ et $P_{i-1}(\lambda_0)$ sont de signe opposé.
4. P_i a exactement *i* racines réelles distinctes qui séparent les *i+1* racines de P_{i+1}.

Bissection de Givens

Soit μ un réel. Soit $E(i,\mu)=\{+, signe(P_1(\mu))\ldots signe(P_i(\mu))\}$. On définit $N(i,\mu)$ le nombre de changement de signe dans $E(i,\mu)$. Alors $N(i,\mu)$ représente le nombre de racines de P_i qui sont inférieures et distinctes de μ.

On ordonne les λ_i de manière croissante : $\lambda_1 < \cdots < \lambda_n$. Considérons un intervalle $[a_0,b_0]$ contenant λ_i. Soit $\mu = \frac{a_0+b_0}{2}$ (bissection). On calcule $N(n,\mu)$. Si $N(n,\mu) > i$, alors $\lambda_i \in [a_0,\mu]$ sinon $\lambda_i \in [\mu,b_0]$. On réitère…

Une variante pour éviter les overflows

(…)

Méthode de la puissance itérée

Soit *A* une matrice quelconque, éventuellement à coefficient complexe.

Théorème

Soit *A* diagonalisable dont la valeur propre λ_1 (de plus grand module) est unique. Soit q_0 un vecteur non orthogonal au sous-espace propre à gauche[2] associé à λ, tel que $\|q_0\|=1$. On suppose $x_{n+1} = Aq_n$, $q_{n+1} = \frac{x_{n+1}}{\|x_{n+1}\|}$. Alors, on a : $\left(\frac{\overline{\lambda_1}}{|\lambda_1|}\right)^k q_k \to \vec{u_1}$ le vecteur propre associé à λ_1 et $\frac{x_{k+1}(j)}{q_k(j)} \to \lambda_1$ pour tout *j* tel que $q_k(j) \neq 0$.

En pratique !

- En pratique l'hypothèse *"q_0 un vecteur non orthogonal au sous-espace propre à gauche[3] associé à λ"* n'est pas nécessaire, car le phénomène d'arrondi permet de s'éloigner du cas orthogonal !

---

[2] vecteur propre à gauche pour λ si $A^* v = \overline{\lambda} v$

[3] vecteur propre à gauche pour λ si $A^* v = \overline{\lambda} v$

- Si A n'est pas diagonalisable, la méthode fonctionne encore, mais la convergence est plus lente.
- En pratique, on choisit la norme infinie, et $q_0 = (1,\ldots,1)$.

Méthode de la déflation

On suppose maintenant que l'on connaît λ_1 et u_1, avec $|\lambda_1| > \cdots > |\lambda_n|$ et la base de vecteur propre $(u_1,\cdots,u_n)$.
On cherche λ_2 et u_2.
On suppose $A = A^*$ (A symétrique ?). Le principe est simple, on forme la matrice *B* tel que $B = A - \lambda_1 u_1 u_1^*$.
Cette matrice possède les mêmes valeurs propres, les mêmes vecteurs propres, sauf λ_1 et u_1. Par conséquent, si l'on applique la méthode de la puissance itérée, on va obtenir λ_2 et u_2.

En pratique, on ne calcule pas *B*… La seule différence avec la méthode de la puissance itérée est dans la formule $x_{k+1} = A(q_k - \alpha_k u_1)$. (Phénomène d'arrondi compensatoire inverse…)

Cette méthode permet en réitérant de trouver λ_3, λ_4 mais pas au delà, car il y a répercussion des erreurs d'arrondi.

Méthode de la puissance itérée inverse

Cette méthode permet de calculer la plus petite valeur propre λ_n et son vecteur propre u_n. On prend $B = A^{-1}$ et on applique la méthode de la puissance itérée à B, ce qui donne u_n et $\frac{1}{\lambda_n}$.

Pour trouver la valeur propre la plus proche de μ, on applique la méthode de la puissance itérée à $B = A - \mu.Id$, ce qui donne u_k et $\lambda_k - \mu$.

Approximation polynomiale & Intégration numérique

Approximations polynomiales

Polynômes orthogonaux

Soit $\langle P,Q\rangle = \int_a^b P(x)Q(x)W(x)dx$ avec $W(x) \geq 0$ un poids. Considérons $P_0, P_1, \cdots, P_n$ avec P_i un polynôme de degrés *i*. Alors $\langle P_i, P_j \rangle = 0$ pour $i \neq j$.

Legendre

- $[-1,1]$, $W(x) = 1$, $\langle P,Q\rangle = \int_{-1}^{1} P(x)Q(x)dx$.
- $P_0 = 1$, $P_1(x) = x$, $P_n(x) = \frac{2n-1}{n} xP_{n-1}(x) - \frac{n-1}{n} P_{n-2}(x)$.
- $\|P_n\|^2 = \frac{2}{2n+1}$.

Laguerre

- $[0,+\infty[$, $W(x) = e^{-x}$, $\langle P,Q\rangle = \int_0^{+\infty} P(x)Q(x)e^{-x}dx$.
- $L_0 = 1$, $L_1(x) = -x+1$, $L_n(x) = (2n-x-1)L_{n-1}(x) - (n-1)^2 L_{n-2}(x)$
- $\|P_n\|^2 = (n!)^2$.

Hermite

- $\langle P,Q\rangle = \int_{-\infty}^{+\infty} P(x)Q(x)e^{-x^2}\,dx$.
- $H_0 = 1$, $H_1(x) = 2x$, $H_n(x) = 2xP_{n-1}(x) - 2(n-1)H_{n-2}(x)$.
- $\|P_n\|^2 = 2^n n!\sqrt{\pi}$.

Tchebychev

- $\langle P,Q\rangle = \int_{-\infty}^{+\infty} \frac{P(x)Q(x)}{\sqrt{1-x^2}}\,dx$.
- $T_0 = 1$, $T_1(x) = x$, $T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x)$.
- formule explicite : $T_n(x) = \begin{cases} \cos(n\cos^{-1}(x)), |x| \le 1 \\ ch(nch^{-1}(x)), |x| \ge 1 \end{cases}$. T_n a *n* zéros, $x_k = \cos\left(\frac{2k-1}{n}.\frac{\pi}{2}\right) \in [-1,1]$ pour $1 \le k \le n$ et *n+1* extrema sur $x'_k = \cos\left(\frac{k+1}{n}.\right) \in [-1,1]$.
- $\|P_0\|^2 = \pi$, $\|P_n\|^2 = \frac{\pi}{2}$.

Théorèmes

- Soit $E_n = \{P / \deg(P) = n \text{ et } a_n = 1\}$. Alors $\forall P \in E_n, \sup_{[-1,1]} \left|\frac{T_n(x)}{2^{n-1}}\right| \le \sup_{[-1,1]} |P(x)|$.
- Soit $F_n = \{P / \deg(P) = n \text{ et } P(\alpha) = \beta\}$ avec $\alpha \notin [-1,1]$ et β fixé. Alors $\forall P \in F_n$, $\sup_{[-1,1]} \left|\frac{\beta T_n(x)}{T_n(\alpha)}\right| \le \sup_{[-1,1]} |P(x)|$.
- Soit $G_n = \{P / \deg(P) = n \text{ et } P(\alpha) = \beta\}$ avec $\alpha \notin [a,b]$ et β fixé. Alors $\forall P \in G_n$, $\sup_{[a,b]} \left|\frac{\beta.T_n\left(\frac{a+b-2x}{a-b}\right)}{T_n\left(\frac{a+b-2\alpha}{a-b}\right)}\right| \le \sup_{[a,b]} |P(x)|$.

Algorithme de *Remes*

Soient un intervalle $[a,b]$, et un fonction $f \in C([a,b],R)$. $\|f\|_\infty = \sup_{[a,b]} |f(x)|$. On cherche un polynôme *P* de degrés inférieur ou égal à *n* tel que $\|P - f\|_\infty \le \|Q - f\|_\infty$ pour tout polynôme *Q* de degrés inférieur ou égal à *n*.

Théorème

On considère $x_0 < x_1 < \cdots < x_{n+1}$ *n+2* points de $[a,b]$.

1. $\exists! P / \deg(P) \le n \text{ et } \forall i,\ f(x_i) - P(x_i) = (-1)^i (f(x_0) - P(x_0))$ (equioscillations)
2. $\max_{0\le i\le n+1} |f(x_i) - P(x_i)| \le \max_{0\le i\le n+1} |f(x_i) - Q(x_i)|$ pour tout *Q* tel que $\deg(Q) \le n$, c'est-à-dire que *P* est la meilleure approximation pour les points x_i, mais pas meilleure approximation uniforme.

Corollaire

Posons $\alpha = |f(x_i) - P(x_i)|, \forall i$. Si $\alpha = \|f - P\|_\infty$ alors $\|f - P\|_\infty \leq \|f - Q\|_\infty$ pour tout polynôme *Q* de degrés *n*. *P* est la meilleure approximation uniforme sur $[a,b]$.

Algorithme

Du point de vue informatique, on se donne une tolérance $\varepsilon > 0$, d'où si $\left|\alpha - \|f - P\|_\infty\right| \leq \varepsilon$, alors pour tout polynôme *Q* de degrés *n* $\|f - P\|_\infty \leq \|f - Q\|_\infty + \varepsilon$ …

1. Choisir *n+2* points.
2. Si condition vrai, alors fin.
3. Sinon, échange : on introduit *y* en respectant le principe d'oscillation… ???
4. On recommence avec la nouvelle famille de points.

On démontre que l'algorithme converge.

Approximation par interpolation

Considérons *n+1* points $x_0 < x_1 < \cdots < x_n$. Soit *P* un polynôme de degrés *n* tel que $P(x_i) = f(x_i)$. Un tel polynôme existe et on démontre qu'il est unique. On définit $W_j(x) = \prod_{\substack{k=0 \\ k \neq j}}^{n} \left(\frac{x - x_k}{x_j - x_k} \right)$ un polynôme de degrés *n*.

On a $W_j(x_i) = \begin{cases} 1 \; si \; i = j \\ 0 \; si \; i \neq j \end{cases}$. Alors $P(x) = \sum_{j=0}^{n} f(x_j) W_j(x)$.

Formule de *Gregory - Newton*

Soit *h* le pas. $x_k = x_0 + kh$. On définit $\Delta' f(x) = f(x+h) - f(x)$, $\Delta^2 f(x) = \Delta' \Delta' f(x), \ldots$ On commence par calculer les $\Delta^i f(x_0)$ pour $0 \leq i \leq n$. La formule de *Gregory - Newton* définit le polynôme *P* de degrés *n*

$P(x) = f(x_0) + \binom{u}{1} \Delta' f(x_0) + \ldots + \binom{u}{n} \Delta^n f(x_0)$ avec $u = \frac{x - x_0}{h}$ et $\binom{u}{i} = \frac{1}{i!} u(u-1) \cdots (u-i+1)$.

Remarquons que si $x = x_k, u = k$ et $\binom{u}{i} = C_k^i$. On estime l'erreur $|f(x) - P(x)| \leq \frac{h^{n+1}}{4(n+1)} M_{n+1}$ avec $M_{n+1} = \|f^{(n+1)}(x)\|_\infty$. P_n ne converge pas vers *f* à cause des M_n. Aussi pour avoir la convergence uniforme sur $[a,b]$, on réalise l'interpolation sur *N* sous-segments de longueur $\frac{b-a}{N}$ avec *5* ou 6 points (en pratique).

Intégration numérique

Si P_n converge uniformément vers *f* sur $[a,b]$, alors $\int_a^b P_n$ tend vers $I = \int_b^a f$. On partage l'intervalle $[a,b]$ en *N* sous-segments $[x_i, x_{i+1}]$ de longueur $h = \frac{b-a}{N}$, avec $x_0 = a$ et $x_N = b$.

Stabilité

Les formules sont de la forme $\sum_i a_i f(x_i)$ avec nécessairement $\sum_i a_i = b - a$.

La méthode est stable si pour tout *h*, il existe *A* tel que $\left|\sum_i a_i f(x_i) - \sum_i a_i f(x_i + \varepsilon_i)\right| < A\varepsilon$, avec $\varepsilon = \sup_i |\varepsilon_i|$.

Théorème : Si $a_i \geq 0$ alors la formule est stable.

Formule des rectangles

Sur chaque sous-segment $[x_i, x_{i+1}]$, on approche f avec un polynôme de degrés 0, c'est-à-dire par une constante égale à la valeur milieu. Donc $R_h = h\sum_{i=0}^{N} f\left(x_i + \frac{h}{2}\right)$. On estime l'erreur $|I - R_h| \leq \frac{(b-a)}{24} M_2 h^2$. Cette méthode est d'ordre 2, et non d'ordre 1 comme on le croit souvent !

Formule des trapèzes

Les polynômes qui interpolent sont de degrès 1, ce sont des droites. $T_h = h\left[\frac{f(x_0)}{2} + \frac{f(x_N)}{2} + \sum_{i=1}^{N-1} f(x_i)\right]$.

On estime l'erreur $|I - T_h| \leq \frac{(b-a)}{12} M_2 h^2$. Cette méthode est d'ordre 2 et n'est pas meilleur que celle des rectangles (sinon moins bonne).

Formule de *Simpson*

$S_h = \frac{h}{6}\left\{f(x_0) + f(x_N) + \sum_{i=0}^{N-1} 2f(x_i) + 4f\left(x_i + \frac{h}{2}\right)\right\}$. La méthode est d'ordre 4.

Spline

Théorème

Soi $k \geq 1$. Il existe une et une seule fonction Γ sur $[x_1, x_N]$ tel que $\Gamma(x_i) = y_i$. Sur chaque segment $[x_i, x_{i+1}]$, Γ est un polynôme de degrés *2k-1* avec $\Gamma^{(2k-2)}(x_i)$ qui existe et $\Gamma^{(2k-2)}(x_1) = \Gamma^{(2k-2)}(x_N) = 0$, de telle sorte donc que Γ soit de classe *2k-2* sur $[x_1, x_N]$.

Spline cubique (*k = 2)*

(…)

Équations différentielles

Généralités

Problème de *Cauchy*

Soit $f : [x_0, x_0 + a] \times R^M \to R^M$, continue, telle que $(x, y) \mapsto f(x, y)$. Etant donnée une condition initiale y_0, on cherche $y : [x_0, x_0 + a] \to R^M$ de classe C^1 telle que $\begin{cases} y'(x) = f(x, y(x)) \\ y(x_0) = y_0 \end{cases}$, problème de *Cauchy* dimension *M* et d'ordre 1.

Le problème de *Cauchy* de dimension *M* et d'ordre *P* : $\begin{cases} y^{(P)}(x) = g\left(x, y, y', \cdots, y^{(P-1)}\right) \\ y(x_0) = y_0,\ y'(x_0) = y'_0,\ \cdots,\ y^{(P-1)}(x_0) = y_0^{(P-1)} \end{cases}$.

On se ramène au problème de *Cauchy* d'ordre 1 en posant $z = \begin{pmatrix} y \\ y' \\ \vdots \\ y^{(P-1)} \end{pmatrix}$, de dimension PM.

Théorème (K - lipschitzienne)

Si il existe *K* tel que $\forall x, y_1, y_2, \left\| f(x, y_1) - f(x, y_2) \right\| \le K \left\| y_1 - y_2 \right\|$, alors le problème de *Cauchy* d'ordre 1 possède une solution unique.

Méthode de résolution numérique

On se place dans le cadre de ce théorème, et l'on cherche à calculer une solution. On divise l'intervalle $[x_0, x_0 + a]$ en *N* de telle sorte que $x_N = x_0 + a$; $x_n = x_0 + nh$ avec le pas $h = \frac{a}{N}$. On voudrait que les y_n soient proches de $y(x_n)$. La formule générale employée pour la résolution numérique des équations différentielles est $\begin{cases} y_{n+1} = y_n + h\Phi(x_n, y_n, h) \\ y_0 \end{cases}$.

Stabilité

On introduit une perturbation, $\begin{cases} z_{n+1} = z_n + h\Phi(x_n, z_n, h) + h\varepsilon_n \\ z_0 = y_0 \end{cases}$, et l'on observe le comportement de la méthode. Posons $\varepsilon = \sup_n \left\| \varepsilon_n \right\|$. La méthode est stable si et seulement si $\exists A / \max_n \left\| z_n - y_n \right\| \le A\varepsilon$.

Consistance

La méthode est consistante si et seulement si $\max_n \left\| \frac{y(x_{n+1}) - y(x_n)}{h} - \Phi(x_n, y(x_n), h) \right\| \xrightarrow[h \to 0]{} 0$. La méthode est consistante d'ordre *p* si et seulement si $\exists A / \max_n \left\| \frac{y(x_{n+1}) - y(x_n)}{h} - \Phi(x_n, y(x_n), h) \right\| \le Ah^p$.

Convergence

La méthode est convergente si et seulement si $\max_n \left\| y_n - y(x_n) \right\| \xrightarrow[h \to 0]{} 0$. Il y a convergence d'ordre *p* si et seulement si $\exists A / \max_n \left\| y_n - y(x_n) \right\| \le Ah^p$.

Si la méthode est stable et consistante [d'ordre *p*], alors la méthode est convergente [d'ordre *p*].

Théorème stabilité

Si il existe *K* tel que $\forall x, y_1, y_2, \left\| \Phi(x, y_1, h) - \Phi(x, y_2, h) \right\| \le K \left\| y_1 - y_2 \right\|$, c'est-à-dire si ϕ est *K* - lipschitzienne, alors la méthode est stable.

Théorème consistance

- Si $\Phi(x, y, 0) = f(x, y)$, on a consistance d'ordre 1.

- Si de plus, $\left(\frac{\partial \Phi}{\partial h}\right)(x,y,0) = \frac{1}{2}\left(\frac{\partial f}{\partial x}\right)(x,y)$, alors on a consistance d'ordre 2.
- Si de plus, $\left(\frac{\partial^2 \Phi}{\partial h^2}\right)(x,y,0) = \frac{1}{3}\left(\frac{\partial^2 f}{\partial x^2}\right)(x,y)$, alors on a consistance d'ordre 3.
- Etc. …

Méthode de *Euler*

$\Phi(x,y,h) = f(x,y)$, ordre 1, stabilité acquise (ϕ et f lipschitzienne). Intuitivement, on traduit la formule du taux d'accroissement $f(x_n, y_n) = \frac{y_{n+1} - y_n}{h}$. La formule sera donc $y_{n+1} = y_n + hf(x_n, y_n)$.

Méthode du développement de *Taylor*

$\Phi(x,y,h) = f(x,y) + \frac{1}{2}h\left(\frac{\partial f}{\partial x}\right)(x,y) + \frac{1}{3}\frac{h^2}{2!}\left(\frac{\partial f}{\partial x}\right)(x,y) + \ldots$, consistante de l'ordre que l'on veut, stabilité acquise. Mais en pratique les calculs de dérivées rend la méthode impraticable !

Méthode de *Runge - Kutta*

Ordre 2

On cherche $\Phi(x,y,h) = \beta f(x,y) + \alpha f(x + \lambda h, y + \mu h f(x,y))$. L'ordre 1 impose $\alpha + \beta = 1$ et l'ordre 2 impose $\lambda = \mu \; et \; \alpha\lambda = \alpha\mu = 1/2$.

- Si $\alpha = 1$, alors $\beta = 0 \; et \lambda = \mu = 1/2$, d'où la formule $y_{n+1} = y_n + hf\left(x_n + \frac{h}{2}, y_n + \frac{h}{2} f(x_n, y_n)\right)$.

  Soit le prédicteur $y_{n+1/2}$, la méthode s'écrit : $\begin{cases} y_{n+1/2} = y_n + \frac{h}{2} f(x_n, y_n) \\ y_{n+1} = y_n + hf(x_n + h/2, y_{n+1/2}) \end{cases}$ La méthode est d'ordre 2, et correspond à une amélioration de *Euler* : le taux d'accroissement est calculé pour le point milieu $(x_n + h/2, y_{n+1/2})$.

- Si $\alpha = 1/2$ …

Ordre 4

- $\Phi(x,y,h) = \frac{1}{6}(P_1 + 2P_2 + 2P_3 + P_4)$
- $P_1 = f(x,y)$
- $P_2 = f(x + h/2, y + h/2.\mathrm{P}_1)$
- $P_3 = f(x + h/2, y + h/2.\mathrm{P}_2)$
- $P_4 = f(x + h, y + h.\mathrm{P}_3)$

Choix de *h*

On se donne une tolérance $\varepsilon > 0$.

- On choisit arbitrairement *h* et on applique la méthode.

- On recommence avec $h/2$. Si le résultat obtenu pour un pas divisé par 2, est à ε près celui obtenu pour un pas de *h* alors le pas est bon, sinon on réitère. Attention il faut comparer $x_n = x_0 + nh$ et $x_{2n} = x_0 + 2n.\frac{h}{2}$.

Méthode à pas multiples
(…)

Équations différentielles du 2ème ordre (avec conditions aux bords)

Problème de *Dirichlet*

Méthode des différences finies

Méthode des éléments finis

Transformée de *Fourier* discrète

Algorithme de TFD rapide
(…)

Application
(…)