PROJET C++
Différences finies pour le laplacien.

1 La classe MatriceBande

1.1 La méthode de Gauss.

On a établi en cours algèbre matriciel que la matrice A inversible peut s'écrire comme le produit d'une matrice triangulaire inférieure et d'une matrice triangulaire supérieure: $M = L \times U$.

Le système à résoudre [A]{X} = {B} devient:

[L]{Y} = {B} Descente

[U]{X} = {Y} Remontée

Pour déterminer les matrices L et U stockées sous la forme d'une seule matrice où L est sa partie inférieure et U sa partie supérieure et qui de plus écrase la matrice initiale, on établit les relations suivantes:

Pour $k = 0, N-1$

Pour $i = k+1, N-1$

$C = \dfrac{A_{\mathrm{ik}}}{A_{\mathrm{kk}}}$

$A_{\mathrm{ik}} = C$

Pour j=k+1,N-1

$A_{\mathrm{ij}} = A_{\mathrm{ij}} - C.A_{\mathrm{kj}}$

Pour résoudre le système, il faut effectuer une descente:

Pour $i = 0, N-1$

$X_i = B_i - \sum_{k=0}^{k=i-1} A_{\mathrm{ik}} \times X_k$

et une remontée:

Pour $i = N-1, 0$

$X_i = \left(X_i - \sum_{k=i+1}^{k=N-1} A_{\mathrm{ik}} \times X_k \right) / A_{\mathrm{ii}}$

1.2 Matrice bande.

Définition 1.

La matrice M est une matrice bande de largeur d $\Leftrightarrow M_{i,j} \neq 0$ pour $j = i - d$ et $j = i + d$, $M_{i,j} = 0$ *pour* $j < i - d$ *et* $j > i + d$

Exemples:

$$M_1 = \begin{pmatrix} -2,1,0,0,0,0,0,0 \\ 1,-2,1,0,0,0,0,0 \\ 0,1,-2,1,0,0,0,0 \\ 0,0,1,-2,1,0,0,0 \\ 0,0,0,1,-2,1,0,0 \\ 0,0,0,0,1,-2,1,0 \\ 0,0,0,0,0,0,1,-2 \end{pmatrix} \text{ avec } d=1$$

$$M_2 = \begin{pmatrix} \alpha,1,0,1,0,0,0,0,0 \\ 1,\alpha,1,0,1,0,0,0,0 \\ 0,1,\alpha,1,0,1,0,0,0 \\ 1,0,1,\alpha,1,0,1,0,0 \\ 0,1,0,1,\alpha,1,0,1,0 \\ 0,0,1,0,1,\alpha,1,0,1 \\ 0,0,0,1,0,1,\alpha,1,0 \\ 0,0,0,0,1,0,1,\alpha,1 \\ 0,0,0,0,0,1,0,1,\alpha \end{pmatrix} \text{ avec } d=3$$

La matrice M est stockée dans un tableau uni-dimensionnel A de taille (2 d+1)n et l'adresse du coefficient M_{ij} est (d + i - j)n +j dans A.

L'adaptation du programme de l'algorithme de Gauss dans le cas où la matrice est bande peut s'écrire:

boucle pour i = 0 , n-1
  boucle pour k =i+1, Minimun(n-1,i+d)
    $C = \frac{A_{ki}}{A_{ii}}$
    $A_{ki} = C$
      boucle pour j = i+1,Minimun(n-1,i+d)
        $A_{k,j}$= $A_{k,j}$- C * $A_{i,j}$
      fin de boucle j
  fin de boucle k
fin de boucle i

Descente:

boucle pour i = 0 , n-1
  s = 0.
  boucle pour k= Maximum(0,i-d) , Minimun(n-1,i-1)
  s += A_{ik} * x_k
  fin de boucle k
x_i = b_i - s
fin de boucle i

Remontée:
boucle pour i = n-1,0
s = 0
  boucle pour k = i+1, Minimum(n-1,i+d)
  s += A_{ik} * x_k
  fin de boucle k
x_i = (x_i - s) / A_{ii}
fin de boucle i

1.3 La classe Matrice Bande

Après avoir défini la classe Vecteur et ses opérateurs, la classe MatriceBande est définie par:

Attributs privés:

- double * A ; // Coefficients de la matrice.
- int N ; // Dimension de la matrice
- int D ; // largeur de bande
- int L ; // Taille de A : (2*D+1)*N

Méthodes publiques:

1. MatriceBande(int n , int d) // Constructeur arguments: n = dimension et d = largeur de bande
2. MatriceBande(const MatriceBande & M) // Constructeur par recopie.
3. ~MatriceBande() // Destructeur
4. MatriceBande & operator = (const MatriceBande & M) // Opérateur d'affectation
5. double & operator (int i , int j) // retourne le coefficient ij de la matrice
6. Vecteur operator * (const Vecteur &) // Multiplication de la matrice et d'un vecteur.
7. void Factorise() // Effectue la factorisation de gauss.
8. void Resoudre (Vecteur &) // Effectue la résolution du système.
9. void Affiche () // Affichage de la matrice.

Méthodes privées:

int Indice (int i , int j) // return A[N*(D+i-j)+j]

void Initialise (double * B) // initialise les coefficients A de la matrice avec le tableau B
// si B est NULL alors les coefficients sont nuls.

void Copy(const MatriceBande & M) // Copie de la matrice donnée M et pour éviter
// la répétition d'instructions pour le constructeur de recopie et opérateur d'affectation

On pourra tester le composant avec la matrice M1 définie précédemment et le vecteur second membre V(1,1,1,....1)

2 Différences finies pour l'opérateur de Laplace.

On se propose de résoudre le problème suivant:

$$-\sum_{i=1}^{n} \frac{\partial^2 u}{\partial x_i^2} = f \text{ sur } [0,1]^n \text{ et } u = g \text{ sur les bords avec n=1,2}$$

f et g possèdent une régularité suffisante pour que le problème possède une solution unique dans le bon espace fonctionnel (voir cours Analyse Fonctionnelle ou Equations aux dérivées partielles ou Mr John Cagnol et Mme Bérengère Branchet)

2.1 Dérivées partielles.

2.1.1 Approximation de la dérivée première et seconde d'une fonction réelle.

Considérons le développement en séries de Taylor d'une fonction u(x) de R dans R.

$u(x+h) = u(x) + hu'(x) + \frac{h^2}{2!}u''(x) + \frac{h^3}{3!}u^{(3)}(x) + ...$

$u(x-h) = u(x) - hu'(x) + \frac{h^2}{2!}u''(x) - \frac{h^3}{3!}u^{(3)}(x) + ...$

Le développement d'ordre 1 donne une approximation de la dérivée première:

$$u'(x) = \frac{u(x+h) - u(x-h)}{2h}$$

Le développement d'ordre 2 donne une approximation de la dérivée seconde:

$$u''(x) = \frac{u(x+h) - 2\,u(x) + u(x-h)}{h^2}$$

2.1.2 Approximation des dérivées partielles d'une fonction de $\mathbf{R^2}$ dans R.

Considérons les développements en séries de Taylor d'une fonction u(x,y) en x et en y:

$$u(x+h,y) = u(x,y) + h\frac{\partial u(x,y)}{\partial x} + \frac{h^2}{2!}\frac{\partial^2 u(x,y)}{\partial x^2} + \frac{h^3}{3!}\frac{\partial^3 u(x,y)}{\partial x^3} + ...$$

$$u(x-h,y) = u(x,y) - h\frac{\partial u(x,y)}{\partial x} + \frac{h^2}{2!}\frac{\partial^2 u(x,y)}{\partial x^2} - \frac{h^3}{3!}\frac{\partial^3 u(x,y)}{\partial x^3} + ...$$

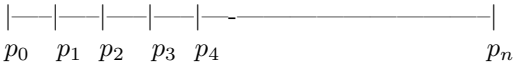
$$u(x,y+h) = u(x,y) + h\frac{\partial u(x,y)}{\partial y} + \frac{h^2}{2!}\frac{\partial^2 u(x,y)}{\partial y^2} + \frac{h^3}{3!}\frac{\partial^3 u(x,y)}{\partial y^3} + ...$$

$$u(x,y-h) = u(x,y) - h\frac{\partial u(x,y)}{\partial y} + \frac{h^2}{2!}\frac{\partial^2 u(x,y)}{\partial y^2} - \frac{h^3}{3!}\frac{\partial^3 u(x,y)}{\partial y^3} + ...$$

$$\frac{\partial^2 u(x,y)}{\partial x^2} + \frac{\partial^2 u(x,y)}{\partial y^2} = \frac{u(x+h,y) + u(x,y+h) - 4u(x,y) + u(x-h,y) + u(x,y-h)}{h^2}$$

2.2 La laplace 1D

Considérons le segment $[0,1]$, on définit une subdivision p_i avec $i = 0, n$



L'opérateur discrétisé de la Laplace s'écrit:

$$\frac{-u_{i-1} + 2u_i - u_{i+1}}{h^2} = f_i \qquad \text{i=0,n}$$

où u_i et f_i sont les valeurs aux points p_i des fonctions $u(x)$ et $f(x)$

La Condition de Dirichlet aux extrémités s'écrit:

$u_0 = g(0)$
$u_n = g(1.)$

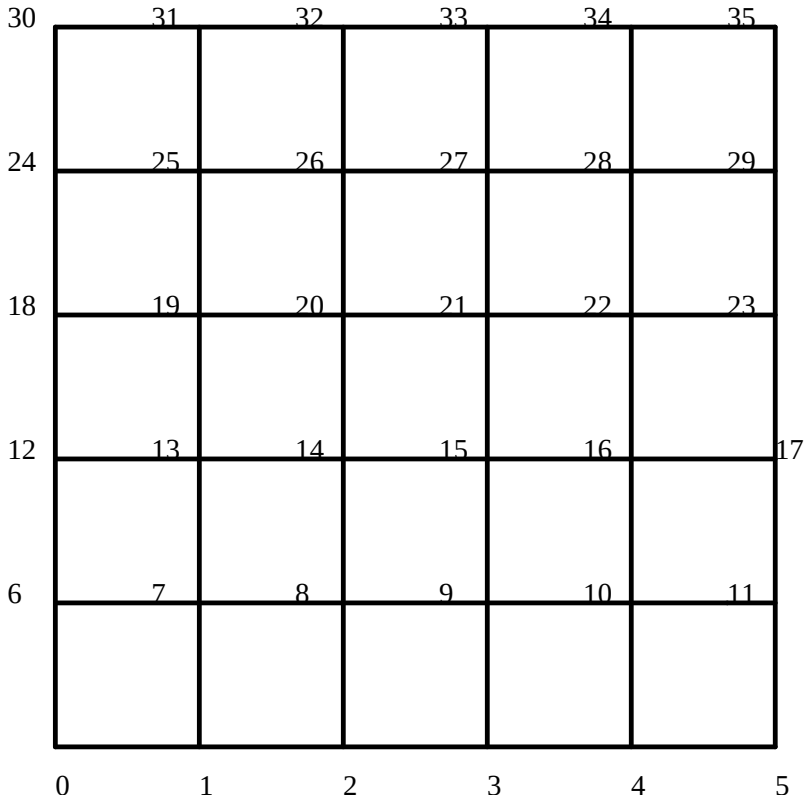
Par conséquent, la ligne et la colonne 0 auront tous les coefficients nuls sauf le coefficient

diagonal $A_{00} = 1$ et le second membre de la ligne 0 est égal à g(0)
de même pour la ligne et la colonne n.
Ecrire la classe Laplace1D qui hérite de MatriceBande et résout ce systéme linéaire avec en données le nombre de subdivisions n de [0,1] et les fonctions f et g.

```
class laplace1D : public MatriceBande
{
   Vecteur Solution;
    int N; // nombre de subdivisions du segments
    double (*F) (double) ; // fonction source
    double (*G)(double) ; // fonction dirichlet
    public:
     Laplace1D ( int =0 , double (*f)(double) = NULL , double (*g)(double)=NULL);
     void SecondMembre(); // Calcule le second membre du systeme
     void InitialiseMatrice() ; // Calcule la matrice
     void Calcul(); // Résout le système.
     double operator [ ] (in t i) const { return Solution[i];}
};
```

2.3 La Laplace 2D

Considérons le carré unité [0,1]x[0,1] où une grille nxn est définie:



L'opérateur discrétisé de la Laplace s'écrit:

$$\frac{-u_{i-1} - u_{i-n} + 4u_i - u_{i+1} - u_{i+n}}{h^2} = f_i \qquad \text{pour i=0,n}^2\text{-1}$$

où u_i et f_i sont les valeurs aux points p_i des fonctions $u(x)$ et $f(x)$
et n le nombre de subdivisions en x et y (dans l'exemple n=6)

Ecrire la classe Laplace2D qui hérite de la classe MatriceBande et résout ce problème avec en données n et les fonctions f et g.

3 La Méthode du Gradient conjugué.

3.1 L'algorithme.

On va tester un autre algorithme de résolution d'un système linéaire (avec une matrice définie positive) qui présente sous certaines conditions des avantages par rapport à celui de Gauss.

L'algorithme:

Soit à résoudre le système:

$\mathrm{Mx} = b$

On se donne une solution initiale x_0

On définit les vecteurs initiaux suivants:

r_0=b-M.x_0

p_0=r_0

$\beta_0 = 0$

On définitla suite pour i $\geqslant$ 0

$$\alpha_i = \frac{\| \mathrm{r}_i \|^2}{\mathrm{Mp}_i . p_i}$$

x_{i+1}=$x_i + \alpha_i p_i$

r_{i+1}=$r_i - \alpha_i \mathrm{Mp}_i$

p_{i+1}=$r_{i+1} + \beta_i p_i$

$$\beta_{i+1} = \frac{\| \mathrm{r}_{i+1} \|^2}{\| \mathrm{r}_i \|^2}$$

La suite (x_i) converge en moins n itérations.

En fait, il suffit d'ajouter un test de convergence

$\| \mathrm{r}_i \|^2 \leqslant \varepsilon^2$ où ε est très petit .

Ajouter une Méthode GradientConjugue à la classe MatriceBande où cet algorithme est programmé.

Vecteur GradientConjugue(double epsilon) et tester cette méthode sur notre problème.

3.2 Stockage morse.

L'algorithme du gradient conjugué utilise seulement la multiplication de la matrice par un vecteur, ceci nous permet de stocker uniquement les termes non nuls de la matrice, ce qui constitue un grand avantage lorsque notre système à résoudre est de très grande taille.

Considérons notre problème de Laplace:

Sur une ligne i de la matrice M_{ij} seules souvent, les colonnes, i-n,i-1,i,i+1,i+n sont non nulles, mais plus exactement:

Pour i = 0, seules les colonnes i, i+1 et i+n sont non nulles.
Pour i variant de 1 à n-1, seules les colonnes i-1, i, i+1 et i+n sont non nulles.
Pour i variant de n à n^2 -n-1, seules les colonnes i-n,i-1, i, i+1 et i+n sont non nulles.
Pour i variant de n^2 -n à n^2-2 , seules les colonnes i-n, i-1, i et i+1 sont non nulles.
Pour i = n^2-1, seules les colonnes i-n, i-1 et i sont non nulles.
Par conséquent, stocker la matrice en termes non nuls, appelé stockage morse, consiste à stocker pour chaque ligne de la matrice, le nombre de colonnes non nulles, le numéro des colonnes non nulles et le coefficient M_{ij} correspondant en commençant par le terme diagonal.

exemples:
ligne 0 -> 3 colonnes non nulles: 0, 1, n. coefficients:{4.,-1.,-1.}
ligne 1 -> 4 colonnes non nulles: 1, 0, 2, n. coefficients:{4.,-1.,-1.,-1.}
ligne n -> 5 colonnes non nulles: n, 0, n-1, n+1, 2n. coefficients:{4.,-1.,-1.,-1.,-1.}
ligne n+1 -> 5 colonnes non nulles:n+1, 1, n, n+2, 2n+1. coefficients:{4.,-1.,-1.,1-.,-1.}

La classe MatriceMorse est définie par:

Attributs privés:

- double * A ; // Coefficients de la matrice.
- int N ; // Dimension de la matrice
- int L ; // Taille de A
- int * NombreDeColonnesNonNulles // Tableau qui contient pour chaque ligne, le nombre // de colonnes non nulles.
- int * ColonnesNonNulles // Tableau qui contient pour chaque ligne, les // numeros des colonnes non nulles.
- int * PointeurLignes // Tableau qui contient l'adresse des termes // diagonaux de la matrice. // PointeurLignes[0] = 0 // Pour chaque ligne i: // PointeurLignes[i]+=PointeurLignes[i-1]+ // NombreColonnesNonNulles[i-1]

Méthodes privées:

```
void Alloue();
void Delete();
void Initialise( double *);
 void CalculeTaille();
void CalculPointeurLignes();
```

Méthodes publiques:

1. MatriceMorse(int , int *, int *) // Les arguments: la dimension de la matrice, // les tableaux NombreDeColonnesNonNulles et // ColonnesNonNulles et alloue la matrice.
2. MatriceMorse(const MatriceMorse & M) // Constructeur par recopie.
3. ~MatriceMorse() // Destructeur
4. MatriceMorse & operator = (const MatriceMorse & M) // Opérateur d'affectation

5. Vecteur operator * (const Vecteur &) // Multiplication de la matrice et d'un vecteur.
6. void GradientConjugue() // L algorithme du gradient conjugue
7. void Affiche () // Affichage de la matrice.
8. void SetNombreColonnesNonNulles(int *)
9. void SetColonnesNonNulles(int *)

Programmer la classe Laplace2D qui hérite de la classe MatriceMorse, comme précédemment avec la classe MatriceBande.