MM031 - TP4
Méthodes directes de résolution d'un système linéaire

Maxime Chupin : chupin@ann.jussieu.fr

27 Janvier 2016

Exercice 1 : classe matrice

Le but de cet exercice est de programmer une classe matrice carrée dense dans l'optique plus tard de pouvoir résoudre un système linéaire avec une méthode directe. Les valeurs de la matrice seront stockée dans un tableau de `double` *unidimensionnel*. Il faudra donc gérer l'accès aux valeurs avec ce dont on a l'habitude, à savoir un couple d'entiers (i, j) pour accéder à l'élément en i-ème ligne et j–-ème colonne.

1. On part de la classe qui contient la taille ainsi que les données :

```
class matrice
{
private :
  int taille;
  double* data;
};
```

2. Ajouter deux constructeurs dans la classe matrice :
   → un constructeur par défaut qui initialisera la matrice vide c'est-à-dire de taille nulle et avec un tableau de données `NULL`.
   → un constructeur qui prend comme argument un `int` et qui alloue une matrice de cette taille au carré dont tous les coefficients sont nuls ;
   → un constructeur par recopie qui a obligatoirement le prototype suivant.

```
matrice(const matrice& );
```

3. Ajouter un destructeur à la classe qui desalloue la matrice.
4. Ajouter une méthode `int matrice::size()const` qui renvoie la taille de la matrice.
5. Ajouter une méthode `void matrice::affiche()const` qui affiche la matrice.
6. Ajouter deux opérateurs d'accès :

```
double & operator()(int i, int j);
double operator() const (int i, int j);
```

   Le premier permet de modifier le coefficient d'une matrice et le second permet seulement de le récupérer.
7. Surcharger l'opérateur =.

Exercice 2 : classe vecteur

Il s'agit ici de reproduire le travail de l'exercice précédent pour une classe `vecteur`.

1. On part de la classe qui contient la taille ainsi que les données toujours sous la forme (sans doute plus intuitive pour un vecteur) :

```
class vecteur
{
private :
  int taille;
  double* data;
};
```

2. Ajouter deux constructeurs dans la classe vecteur :
   → un constructeur par défaut qui initialisera la matrice vide c'est-à-dire de taille nulle et avec un tableau de données `NULL`.
   → un constructeur qui prend comme argument un `int` et qui alloue un vecteur de cette taille dont tous les coefficients sont nuls ;
   → un constructeur par recopie qui a obligatoirement le prototype suivant :

```
vecteur(const vecteur& );
```

3. Ajouter un destructeur à la classe qui desalloue le vecteur.
4. Ajouter une méthode `double vecteur::size()const` qui renvoie la taille d'un vecteur.
5. Ajouter une méthode `double vecteur::affiche()const` qui affiche le vecteur.
6. Ajouter deux opérateurs d'accès :

```
double & operator()(int i);
double operator() const (int i);
```

   Le premier permet de modifier le coefficient d'un vecteur et le second permet seulement de le récupérer.
7. Implémenter le constructeur par recopie.
8. Surcharger l'opérateur =.

Exercice 3 : Résolution d'un système avec une matrice triangulaire

En vue de l'implémentation de méthode de décomposition LU et de Cholesky, nous allons mettre en place des méthodes de résolution de systèmes linéaires $Ax = b$ où A est soit triangulaire inférieure soit triangulaire supérieur. En effet dans ces cas très particuliers, les calculs sont évidemment grandement simplifiés.

1. Écrire une fonction qui prend en argument une matrice triangulaire inférieure T et un vecteur y

```
vecteur triang_inf(const matrice T, vecteur y);
```

   et qui renvoie $T^{-1} \cdot y$.
2. Idem pour une matrice triangulaire supérieure

```
vecteur triang_sup(const matrice T, vecteur y);
```