TP n° 3 : Héritage

L'objectif de ce TP est l'implémentation finale de la librairie d'algèbre linéaire (qui sera utilisée pour les projets) permettant de réaliser les opérations standardes entres matrices et vecteurs. Une étude détaillée des différents mécanismes mis en jeu devra permettre de mettre en évidence une hiérarchie de classe : une classe mère *Tableau* de laquelle les classes matrices et vecteurs héritent naturellement.

Vous devrez apporter un soin tout particulier à la gestion de la mémoire et à la documentation de votre code.

Les données de la classe mère seront stockées de façon contiguë en mémoire.

La gestion d'erreurs doit se faire via des exceptions (utilisez `throw`).

1. En vous basant sur la classe Vecteur du TP2, implémenter une classe `Darray` contenant les attributs communs des futurs classes filles vecteurs et matrices.

   ```
   class Darray {
   protected:
     int size_; /* taille du tableau */
     double *data; /* données */
     bool owner; /* flag propriétaire de data */
   };
   ```

   Cette classe `Darray` utilise le `double` comme type de base et devra proposer les fonctionnalités suivantes :

   (a) * Les constructeurs/destructeur demandés dans le TP2, l'opérateur `size()` ainsi que les accesseurs `()`

   (b) L'opérateur =

   (c) L'addition, la soustraction, la multiplication et la division par un scalaire.

   (d) L'addition, la soustraction termes à termes et l'opérateur − unaire

   (e) La méthode `view`

   Une fois la question 1 traitée, vérifier que la structure (dossiers, fichiers, nom des classes et fonctions) répond bien aux instructions données : Lancer le script `verifier.py`, celui-ci doit confirmer que la question 1 a été traitée correctement. Relancer `verifier.py` régulièrement (pour chaque question), afin de vérifier que vous respectez bien l'énoncé. **Enfin, le relancer sur l'archive compressée (.tar.gz) avant de l'envoyer.**

2. Construire une classe `Dvector` héritant publiquement de `Darray` qui devra proposer les fonctionnalités suivantes :

   (a) * Les constructeurs/destructeur demandés dans le TP2

   (b) L'opérateur * permettant d'effectuer un produit scalaire entre deux `Dvector`

3. Construire une classe `Dmatrix` (matrices non forcément carrées) héritant publiquement de `Darray`

   ```
   class Dmatrix : public Darray {
   private:
     int m; /* nombre de lignes */
     int n; /* nombre de colonnes */
   };
   ```

Les données de la matrice seront stockées de sorte que `M[i,j] = data[j + i*n]` (stockage ligne). Cette classe devra implémenter les fonctionnalités :

(a) * Les constructeurs/destructeur demandés dans le TP2. Note : prévoir les paramètres pour préciser la dimension de la matrice : `Dmatrix( 2, 3, 4 )` devra créer une matrice de 2 lignes et 3 colonnes, initialisée avec des 4.

(b) * Les opérateurs d'accession `M(line,column)`, `line` et `column` allant de `0` à `n-1`.

(c) Les opérateurs `lines()` et `columns()` permettant d'accéder aux dimensions de la matrice

(d) L'opérateur =

(e) La méthode `line( bool copy, int pos )` permettant l'extraction d'une ligne sous la forme d'une `Dvector` grâce à la méthode `view` (`pos` allant de `0` à `n-1`). Note : Le constructeur par recopie de `Dvector` devra être adapté.

(f) La méthode `column( int pos )` permettant l'extraction d'une colonne sous la forme d'une `Dvector` (cette fonctionnalité ne pourra pas utiliser la méthode `view` car les données d'une colonne ne sont pas stockées de manière contiguë en mémoire).

(g) L'opérateur * permettant d'effectuer un produit matrice*vecteur

(h) L'opérateur * permettant d'effectuer un produit matrice*matrice

(i) La méthode `transpose()` pour transposer une matrice carrée (cette méthode devra transposer la matrice courante et être chaînable)

4. Implémenter la méthode `cholesky()` permettant la factorisation de Cholesky pour les matrices carrées symétriques et définies positives.
   La factorisation sera conservée dans la matrice initiale. On rappelle que la factorisation de Cholesky est donnée par

Algorithm 1 Algorithme de Cholesky

for k=1 à n **do**

$$L_{kk} = \sqrt{A_{kk} - \sum_{s=1}^{k-1} L_{ks}^2}$$

  for i=k+1 à n **do**

$$L_{ik} = \left(A_{ik} - \sum_{s=1}^{k-1} L_{is} L_{ks} \right) / L_{kk}$$

  end for

end for

Sortie (L_{ij})

avec A la matrice initiale (symétrique définie positive) et L une matrice triangulaire inférieure telle que $A = LL^t$.

Note : Pour tester votre algorithme, il est conseillé de générer une matrice L triangulaire inférieure, de calculer $A = LL^t$ (à l'aide des méthodes et opérateurs définis précédement). Ensuite, appliquer cholesky sur A devrait vous faire retomber sur L (aux arrondis près).

Note : En cas d'erreur, on accepte que la matrice ne soit que partiellement factorisée par la fonction.