
TD 5

Vecteurs et matrices

5.1 Objectif

L'objectif de cette séance est de réaliser deux classes représentant (au moins partiellement) les vecteurs et les matrices de l'algèbre linéaire.

Ceci fournit l'occasion d'utiliser à nouveau la classe `vector` de la STL, d'améliorer la maîtrise de la construction et de la copie d'objets complexes, et d'aborder la redéfinition d'opérateurs un peu plus exotiques, comme l'indexation ou l'appel de fonction.

5.2 Vecteurs mathématiques : classe `MVector`

5.2.1 Spécification

Votre mission est de définir une classe `MVector` représentant des vecteurs mathématiques de nombres réels (`double`) avec quelques unes de leurs opérations habituelles. Un tel objet devra être constructible en donnant sa dimension et éventuellement une valeur d'initialisation commune pour toutes les composantes :

```
// vecteur de 10 éléments, tous initialisés à 3.5
MVector mv1(10, 3.5);
// vecteur de 20 éléments, tous initialisés à 0.0
MVector mv2(20);
```

Il doit également exister un constructeur par défaut :

```
MVector mv3; // vecteur de dimension 0
```

Voici une liste (non exhaustive, voir plus bas) des opérations qui doivent être possibles sur les `MVectors` :

- copie à l'initialisation comme à l'affectation, avec ajustement automatique de la dimension;
- addition et soustraction de deux vecteurs de même dimension;
- produit scalaire de deux vecteurs de même dimension;
- comparaison (égalité et inégalité) de deux vecteurs;
- indexation d'un vecteur constant ou non grâce à `operator[]`;
- affichage du contenu d'un vecteur grâce à `operator<<`.

Le fichier `main_MVector.cpp` du répertoire `Linear_Algebra` contient un programme principal de test des fonctionnalités attendues de `MVector`. Il se peut que vous ayez à écrire quelques fonctions supplémentaires par rapport à la liste précédente pour pouvoir exécuter ce programme. Nous vous laissons les déterminer.

Quant une opération apparaît impossible, une exception soigneusement choisie doit être lancée.

5.2.2 Réalisation

Vous réaliserez vos `MVector` en utilisant de manière interne un `vector<double>` de la STL pour contenir les composants.

Vous devez éviter les implémentations inutiles. Essayez au maximum de déléguer les opérations

de `MVector` aux opérations correspondantes de `vector`. N'implémentez pas les fonctions automatiquement fournies par C++ (constructeur par défaut et de copie, affectation de copie, destructeur) lorsque la définition par défaut vous convient.

5.3 Matrices rectangulaires : classe `Matrix`

5.3.1 Spécification

Vous allez utiliser cette classe `MVector` pour réaliser la classe `Matrix` représentant des matrices, tableaux bidimensionnels (rectangulaires) de réels (double) avec quelques unes des opérations habituelles.

Une matrice se construit en passant ses deux dimensions et éventuellement une valeur d'initialisation commune pour toutes les composantes :

```
// une matrice 10x20, toutes composantes égales à 0.0
Matrix mat1(10,20);
// une matrice 5x5, toutes composantes égales à 3.0
Matrix mat2(5, 5, 3.0);
```

Il doit également exister un constructeur par défaut :

```
Matrix mat3; // matrice de dimension 0x0
```

Voici une liste (non exhaustive, voir plus bas) des opérations qui doivent être possibles sur les `Matrix`s :

- copie à l'initialisation comme à l'affectation, avec ajustement automatique de la dimension
- addition et soustraction de deux matrices de mêmes dimensions ;
- produit de deux matrices de dimensions convenables (le nombre de colonnes de la première doit être égal au nombre de lignes de la seconde) ;
- transposition d'une matrice (la méthode `transpose()` ou, si vous le souhaitez, `operator~`, doit retourner une copie transposée de la matrice de départ) ;
- comparaison (égalité et inégalité) de deux matrices ;
- indexation des matrices constantes et non constantes (voir 5.3.3 ci-dessous) ;
- affichage du contenu d'une matrice grâce à `operator<<` ;
- conversion d'un `MVector` en matrice-ligne ;
- éventuellement, conversion d'une `Matrix` en `MVector` pourvu que la matrice de départ soit une matrice-ligne.

Le fichier `main_Matrix.cpp` du répertoire `Linear_Algebra` contient un programme principal de test des fonctionnalités attendues de `Matrix`. Il se peut que vous ayez à écrire quelques fonctions supplémentaires par rapport à la liste précédente pour pouvoir exécuter ce programme. Nous vous laissons les déterminer.

Quant une opération apparaît impossible, une exception soigneusement choisie doit être lancée.

5.3.2 Réalisation

Vous réaliserez vos `MVector` en utilisant de manière interne un vecteur de la STL dont les composantes sont des `MVectors` (`vector<MVector>`).

Vous devez éviter les implémentations inutiles. Essayez au maximum de déléguer les opérations de `Matrix` aux opérations correspondantes de `MVector`. N'implémentez pas les fonctions automatiquement fournies par C++ (constructeur par défaut et de copie, affectation de copie, destructeur) lorsque la définition par défaut vous convient.

5.3.3 Note sur les opérations d'indexation

En C++, il n'est pas possible de définir un opérateur d'indexation `operator[]` qui permette d'indexer une matrice en écrivant `mat[i, j]` (Pourquoi donc, à votre avis ?)

En revanche, l'implémentation proposée devrait vous permettre sans trop de peine d'écrire un opérateur d'indexation autorisant l'écriture `mat[i]`, qui retourne une référence sur la ligne d'indice `i` de la matrice (un `MVector`). Alors l'écriture `mat[i][j]` devient elle aussi légale (et adéquate !), n'est-ce-pas ? Il est même possible d'écrire `mat(i, j)` (noter les parenthèses *rondes*) en définissant l'opérateur *appel de fonction* (dont le nom est, bien évidemment, `operator()`).

Vous devrez également définir deux opérations qui entrent dans la catégorie des indexations :

- l'opération `line()`, telle que `mat.line(i)` retourne une copie de la ligne (le `MVector`) d'indice `i` de la matrice `mat` (noter que c'est différent de `mat[i]` évoqué juste au-dessus) ;
- l'opération `column()`, telle que `mat.column(j)` retourne une copie de la colonne d'indice `j` (un `MVector`) de la matrice `mat`.

5.4 Remarques et conseils

5.4.1 Remarque sur le code fourni

Le code est dans le répertoire `Linear_Algebra`, à l'endroit habituel. Il n'est utilisable que si la hiérarchie établie lors du premier TD est respectée : vérifiez donc que vous avez bien le fichier `default.mk` et le répertoire `include` (avec son contenu !) deux niveaux de répertoires au-dessus de votre répertoire de travail (`../../default.mk` et `../../include`).

Le répertoire `Linear_Algebra` contient le source des deux programmes de test déjà mentionnés (`main_MVector.cpp` et `main_Matrix.cpp` qui donneront les exécutables `tst_MVector.exe` et `tst_Matrix.exe`) ainsi que la trace de leur exécution avec ma propre solution (`tst_MVector.out` et `tst_Matrix.out`).

La `Makefile` elle-même est prête à accueillir votre travail et à le tester avec les deux programmes de test fournis, à condition que vous respectiez le nommage des fichiers : vous devez fournir 4 fichiers sources nommés `MVector.h` (la définition de la classe `MVector`), `MVector.cpp` (son implémentation), `Matrix.h` et `Matrix.cpp` (la même chose pour la classe `Matrix`). Bien entendu, vous devez aussi respecter les noms des classes et des méthodes proposés dans ce sujet. Il suffira alors de faire `make` pour construire les deux exécutables ou encore, par exemple,

```
make tst_Matrix.exe
```

pour construire uniquement le second.

5.4.2 Conseil

L'exercice n'est pas particulièrement court. N'attendez-donc pas d'avoir écrit la totalité d'une classe et de ses méthodes pour commencer à la tester.

Essayer de travailler *incrémentalement*. Définissez les méthodes progressivement dans un ordre compatible avec les programmes de test : par exemple le(s) constructeur(s) est(sont) à définir en premier, puis l'opérateur d'affichage (`operator<<`) qui est indispensable à la macro `OUT` définie dans `../..include/common_defs.h`, etc. Testez au fur et à mesure, en commentant les parties du programme de test correspondant à des fonctionnalités non encore implémentées.