Sorbonne Université
2023–2024

M1 Mathématiques
MU4M056 : Programmation en C++

TP noté 2 : Combinaisons linéaires

L'objectif de ce TP est d'implémenter en C++ des combinaisons linéaires à l'aide des templates. Ces templates seront ensuite testés sur différents cas pratiques.

On veillera à inclure dans chaque fichier toutes les bibliothèques nécessaires et les options de compilation nécessaires. **Il est également impératif de mettre en commentaire, dans tous les fichiers, les nom, prénom et n° d'étudiant.**

1 Introduction

1.1 Description mathématique

Une combinaison linéaire est une expression construite à partir d'un ensemble de termes en multipliant chaque terme par une constante et en ajoutant le resultat. Par exemple, une combinaison linéaire des x_i pour $i \in \{1, \dots, n\}$ serait une expression de la forme

$$L(x) = a_1 x_1 + \cdots + a_i x_i + \cdots + a_n x_n,$$

où les a_i sont des constantes. Les différents termes x_i peuvent représenter des nombres, des vecteurs ou en encore des fonctions.

1.2 Implémentation en C++

L'idée est d'introduire un template de classe,

```
template <typename Value> class LinearCombination;
```

où `Value` est un type arbitraire visant à décrire les constantes a_i de la combinaison linéaire.

Le template de classe est défini de la manière suivante :

```
template <typename Value>
class LinearCombination{
    private:
        int n; // correspond à la taille de la combinaison linéaire
        std::vector<Value> coeff; // correspond aux a_i
    public:
        LinearCombination(int n0=0);
        LinearCombination(const std::vector<Value>&);
        template <class X> X operator()(const std::vector<X>&) const;
        // A COMPLETER
    };
```

2 Questions

2.1 Le template

1. Télécharger les fichiers, écrire vos noms, prénoms et numéros d'étudiant en début des fichiers de code sous forme de commentaires. Compléter tous les fichiers avec les inclusions de fichiers et de bibliothèques nécessaires.

2. Écrire un constructeur qui prend en argument un entier $n0$ et crée une combinaison de taille $n0$ contenant que des 0.

3. Écrire un constructeur qui prend en argument un vecteur v et crée une combinaison linéaire de taille n comprenant les éléments du vecteur v.

4. Ajouter un accesseur à la taille de la combinaison et un accesseur aux éléments de `coeff` à l'aide des crochets `[]` . Il s'agit ici de surcharger `Value operator[] (int i) const` .

5. Ajouter un mutateur aux éléments de `coeff` à l'aide des crochets `[]` . Il s'agit ici de surcharger `Value& operator[] (int i)` .

6. Surcharger l'opérateur `<<` de telle sorte qu'une combinaison linéaire soit écrit avec le format suivant :

$$a_0x_0 + \cdots + a_ix_i + \cdots + a_nx_n,$$

où les a_i doivent être remplacés par les valeurs présentes dans le vecteur `coeff` . Ici, pour les a_i on attend la valeur du coefficient et pour les x_i on attend la chaine de caractère x_i avec le bon i.

7. Surcharger l'opérateur `*` , qui permet de multiplier la combinaison linéaire par un scalaire de même type que les éléments du vecteur `coeff` .

8. Surcharger les opérateurs `+` et `-` qui permettent d'additionner et de soustraire deux combinaisons linéaires de même taille.

Indication : Soit $L_1(x) = a_0x_0+\cdots+a_ix_i+\cdots+a_nx_n$ et $L_2(x) = b_0x_0+\cdots+b_ix_i+\cdots+b_nx_n$, alors on a,

$$L_1(x) + L_2(x) = (a_0 + b_0)x_0 + \cdots + (a_i + b_i)x_i + \cdots + (a_n + b_n)x_n.$$

9. Écrire le template de méthode `operator()` qui permet de calculer $L(x)$ à partir des données du champ privé.

10. Vérifier que le programme `test_combinaison.cpp` compile bien et fournit bien les valeurs correspondantes.

2.2 Combinaisons linéaires de fonctions

On souhaite à présent travailler sur des combinaisons linéaires de fonctions. Le but de cet exercice est de trouver la meilleure combinaison linéaire d'un jeu de données.

Pour cela, on suppose que l'on a plusieurs couples de points (x_i, y_i) et on sait que

$$y_i \approx \alpha \cos(6x_i) + \beta \exp(4x_i).$$

On cherche alors quelle combinaison de paramères (α, β) correspond le mieux à notre jeu de données qui est présent dans le fichier `data.txt` .

11. Écrire dans le fichier `linear_combination.hpp` une fonction

```
std::pair<std::vector<double>,std::vector<double>> ReadData(std::istream& in);
```

qui prend en entrée un fichier contenant sur chaque ligne deux nombres (x_i et y_i) et produit en sortie une paire de deux vecteurs. Dans le premier vecteur, on doit retrouver tous les x_i et dans le second vecteur tous les y_i.

12. Écrire dans le fichier `linear_combination.hpp` le template de méthode suivant

```
template <typename Fonction>
std::vector<double> AppFct(const Fonction& Fct, const std::vector<double>& Pts);
```

qui prend en entrée une fonction f et un vecteur de points x_i et produit en sortie le vecteur $f(x_i)$.

13. Écrire dans le fichier `test_fonction.cpp` un programme qui fait les choses suivantes :

1. Ouvre le fichier `data.txt` et récupère les données des x_i et y_i,
2. Crée les vecteurs $\cos(6x_i)$ et $\exp(4x_i)$. On rappelle que les fonctions cosinus et exponentielle appartiennent à la bibliothèque `cmath` .
3. Calcule la combinaison linéaire

$$L(x_i) = \alpha \cos(6x_i) + \beta \exp(4x_i),$$

, pour trois jeux de paramètres différents,
— $(\alpha, \beta) = (-2.0, 0.1)$
— $(\alpha, \beta) = (2.0, 0.9)$
— $(\alpha, \beta) = (-10.0, 0.2)$

4. Calcule l'erreur

$$\sum_{i=0}^{n} (L(x_i) - y_i)^2$$

pour les trois jeux de paramètres différents et les affiche.

Tester le programme et vérifier que l'erreur la plus petite est bien celle pour $(\alpha, \beta) = (-2.0, 0.1)$.

2.3 Combinaisons linéaires de vecteurs

Pour finir, on souhaite travailler sur des combinaisons linéaires de vecteurs,

$$L(v) = a_1 v_1 + \cdots + a_n v_n,$$

où les a_i sont des scalaires et les v_i des vecteurs. Dans cet exercice, les vecteurs v_i à évaluer seront considérés comme des `std::array` défini dans la bibliothèque `<array>` . Cette structure est très similaire à `std::vector` : on peut accéder aux éléments à l'aide des crochets, on peut connaître la taille à l'aide de la méthode `size()` ... La grande différence pertinente ici est la suivante : au lieu d'un seul type passé en paramètre avec les crochets `<>` , on en a un autre, entier, qui correspond à la taille. Contrairement aux vecteurs, les `std::array` ont une taille qui doit être déterminée à la compilation, et ne peut pas être changée en cours de programme. Ainsi, par exemple, un objet de la classe `std::array<int,5>` est un tableau d'entiers de taille 5.

Le constructeur par défaut crée un `array` de taille n contenant les valeurs par défaut pour le type `Value` . Par exemple, si on écrit dans notre fichier,

```
std::array<int,3> a;
```

l' `array` a est de taille 3 et contient que des 0. De plus, pour construire un `std::array<Value,n>` de taille n avec des valeurs déjà définies, on peut écrire

```
std::array<Value,n> a = {a_1, a_2, ..., a_n};
```

où les a_i sont les n valeurs que l'on veut mettre dans notre `array` . Par exemple, pour mettre le vecteur $\{1, 2, 3\}$ dans un `std::array<int,3>` on écrit,

```
std::array<int,3> a = {1,2,3};
```

Pour commencer, on souhaite tout simplement évaluer des combinaisons linéaires de vecteurs. Pour cela, on a besoin de rajouter des opérations sur les `std::array` pour pouvoir les multiplier à un scalaire, les additionner ou les soustraire entre eux.

14. Surcharger dans le fichier `linear_combination.hpp` les opérateurs `*`, `+`, `-` sur les `template <typename V, unsigned long n> std::array<V,n>`.

15. Vérifier que le programme `test_vecteur.cpp` compile bien et fournit bien les valeurs correspondantes.

À présent, on souhaite pouvoir dire si un vecteur w peut s'écrire comme une combinaison linéaire d'une famille de vecteurs $\{v_1, v_2, \ldots, v_n\}$. Prenant un exemple avec une combinaison linéaire de deux vecteurs de taille $n = 2$. Soient trois vecteurs : $v_1 = (v_{1,1}, v_{2,1})$, $v_2 = (v_{1,2}, v_{2,2})$ et $w = (w_1, w_2)$. On cherche ici à savoir si w peut s'écrire comme une combinaison linéaire de v_1 et v_2 ou en d'autres termes, s'il existe α et β tels que,

$$w = \alpha v_1 + \beta v_2.$$

Cela peut se traduire par le système d'équations suivant :

$$\begin{aligned}(L_1)\ & v_{1,1}\alpha + v_{1,2}\beta = w_1\\ (L_2)\ & v_{2,1}\alpha + v_{2,2}\beta = w_2\end{aligned}$$

On remarque que sur la première ligne (L_1), on obtient une combinaison linéaire avec les premiers éléments de chaque vecteur et sur la deuxième ligne (L_2) on a les deuxièmes éléments de chaque vecteur. Finalement, on obtient deux combinaisons linéaires de scalaires qui correspond à un système de deux équations à deux inconnues (α et β). La résolution d'un système d'équations de ce type peut être fait grâce à l'algorithme du pivot de Gauss donné ci-dessous. On supposera ici, que si le pivot (l'élément $v_{k,k}$) est nul alors il n'existe pas de solution.

Data: $L = \{L_1, \ldots L_n\}$ et w
Result: w
for $1 \leq k \leq n$ **do**
  if $v_{k,k} \neq 0$ **then**
    $L_k \leftarrow \frac{1}{v_{k,k}} L_k$
    $w_k \leftarrow \frac{1}{v_{k,k}} w_k$
    for $1 \leq i \leq n$ *et* $i \neq k$ **do**
      $L_i \leftarrow L_i - v_{i,k} L_k$
      $w_i \leftarrow w_i - v_{i,k} w_k$
    end
  end
  else
    Il n'existe pas de solution
  end
end

Algorithm 1: Pivot de Gauss

16. Écrire dans le fichier `linear_combination.hpp` le template de fonction,

```
template<typename V, unsigned long n>
std::pair<bool,std::array<V>> ResolutionSysteme(
                              std::vector<LinearCombination<V>>,
                              std::array<V,n>);
```

qui implémente l'algorithme du Pivot de Gauss. Cette fonction prend en entrée un vecteur de combinaisons linéaires L_i, un array correspondant au vecteur w et renvoie une paire contenant un booléen indiquant si une solution a été trouvée et la solution si elle existe ou un vecteur nul sinon.

Attention : Dans la suite, on supposera que chaque élément des différents vecteurs doit être considéré comme un `double` .

17. Écrire à la suite du fichier `test_vecteur.cpp` un programme qui fait les choses suivantes :

1. Construit les vecteurs de combinaisons linéaires pour savoir si le vecteur w est une combinaison linéaire de v_1, v_2 et v_3 donc les 3 cas suivants :
   — $v_1 = (1, 2, 4)$, $v_2 = (-1, 1, 3)$, $v_3 = (1, -1, 1)$ et $w = (2, 1, 3)$.
   — $v_1 = (1, 3, 8)$, $v_2 = (2, -1, 2)$, $v_3 = (-3, 2, -2)$ et $w = (-1, 7, 9)$.
   — $v_1 = (1, 3, 1)$, $v_2 = (2, 4, 2)$, $v_3 = (3, 5, -2)$ et $w = (4, 6, -7)$.
   Exemple : Pour le premier cas, on souhaite un vecteur L composé de trois combinaisons linéaires L_1, L_2 et L_3. L_1 est la combinaison linéaire qui contient comme coefficients les premiers éléments de chaque vecteur soit $\{1., -1., 1.\}$. L_2 est la combinaison linéaire qui contient comme coefficients les seconds éléments de chaque vecteur soit $\{2., 1., -1.\}$. Pour finir, L_3 est la combinaison linéaire qui contient comme coefficients les derniers éléments de chaque vecteur soit $\{4., 3., 1.\}$.
2. Résoud le système associé et en déduit, s'ils existent, les coefficients de la combinaison linéaire. Affiche les coefficients trouvés.
3. Pour être sûr que les coefficients trouvés sont corrects, on pourra vérifier que w est bien une combinaison linéaire de v_1, v_2 et v_3 avec les coefficients trouvés (dans le cas où ils existent).

FIN DU SUJET