Sorbonne Université **M1 Mathématiques**
2018–2019 **4M056 : Programmation en C++**

TP noté 1 : Polynômes symétriques et matrices de Vandermonde

L'objectif de ce TP est d'explorer quelques propriétés des matrices de Vandermonde et des polynômes symétriques. 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 de chacun des membres du binôme.**

Ce TP nécessite l'utilisation de la bibliothèque `Eigen` , à toutes fins utiles on rappelle les faits suivants :

— Il faut compiler avec `g++` et l'option `-I /usr/include/eigen3` sur un système Linux (voire l'option `-O2` pour réduire les temps de calculs).

— Il faut déclarer la bibliothèque `<Eigen/Dense>` dans les en-têtes de fichiers.

— Une matrice contenant des entiers et de tailles $N \times M$ (non fixées à la compilation) se déclare par

```
Eigen::Matrix<int, Eigen::Dynamic, Eigen::Dynamic> A(N,M);
```

On rappelle également que pour alléger le code on peut redéfinir (en-dehors du code `main`) le type sous un nom plus compact :

```
typedef Eigen::Matrix<double,Eigen::Dynamic,Eigen::Dynamic> MatrixDouble;
```

— On accède au coefficient (i, j) par `A(i,j)` et la numérotation commence à 0 (et non à 1).

— on additionne (resp. multiplie) des matrices avec l'opérateur `+` (resp. `*`) et on les affecte avec `=` .

1 Préliminaires sur les polynômes symétriques

Un polynôme à N indéterminées $P(X_1, ..., X_N)$ est dit *symétrique* si pour toute permutation $\sigma \in \mathfrak{S}_N$ $P(X_{\sigma(1)}, ..., X_{\sigma(N)}) = P(X_1, ..., X_N)$. Autrement dit, c'est un polynôme invariant par n'importe quelle permutation de ses arguments.

L'espace des polynôme symétriques à coefficients dans un anneau $\mathbb{K}$ est une sous-algèbre de $\mathbb{K}[X_1, ..., X_N]$. Une famille génératrice de cette algèbre est donnée par les *polynômes symétriques élémentaires* définis comme suit :

$$e_p(t_0, ..., t_{k-1}) = \sum_{0 \leq i_1 < i_2 < ... < i_p \leq k-1} t_{i_1} ... t_{i_p}, \forall 0 \leq p \leq k, \quad (1)$$

avec pour convention $e_0 = 1$.

Voci quelques exemples simples pour $N = 3$:

— $e_1(X_1, X_2, X_3) = X_1 + X_2 + X_3$,

— $e_2(X_1, X_2, X_3) = X_1X_2 + X_1X_3 + X_2X_3$,

— $e_3(X_1, X_2, X_3) = X_1X_2X_3$.

Il est possible détablir la relation de récurrence suivante entre les e_p :

$$
\begin{aligned}
e_p(t_0,...,t_{k-1}) &= \sum_{i_p=p-1}^{k-1} t_{i_p} \sum_{0\leq i_1<...<i_{p-1}\leq i_p-1} t_{i_1}...t_{i_{p-1}} \\
&= \sum_{i_p=p-1}^{k-1} t_{i_p} e_{p-1}(t_0,...,t_{i_p-1}).
\end{aligned}
\tag{2}
$$

1. Dans un fichier `symmetric.cpp` , écrire une fonction

```
double elementary_symmetric(const std::vector<double> & t, const int & p);
```

qui repose sur la récurrence (2) et applique e_p au k-uplet des coefficients du vecteur t (où k est la taille de t). Tester la fonction en calculant $e_i(1, 10, 100)$ pour $i = 0, 1, 2, 3$ [1] dans un fichier `test_symmetric.cpp` .

2 La classe `Vandermonde`

Une *matrice de Vandermonde* est une matrice de taille $N \times N$ de la forme

$$
V = \begin{pmatrix}
1 & x_0 & x_0^2 & \cdots & x_0^{N-1} \\
1 & x_1 & x_1^2 & \cdots & x_1^{N-1} \\
1 & x_2 & x_2^2 & \cdots & x_2^{N-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & x_{N-1} & x_{N-1}^2 & \cdots & x_{N-1}^{N-1}
\end{pmatrix},
$$

qui ne dépend que de N nombres (entiers, réels ou complexes) $x_0, ..., x_{N-1}$.

2. Dans un fichier `symmetric.hpp` , déclarer une classe

```
class Vandermonde{
protected:
    std::vector<double> coeff;
    Eigen::Matrix<double,Eigen::Dynamic,Eigen::Dynamic> mat;
    unsigned int nb_coeff;
};
```

dont les champs privés correspondent aux données suivantes :

— `coeff` liste les nombres $x_0, ..., x_N$ sous forme de vecteur ;

— `nb_coeff` renvoie N ;

— `mat` contient la matrice de Vandermonde associée à $x_0, ..., x_{N-1}$.

Tous les codes de la classes seront à écrire dans le fichier `symmetric.cpp` et pas dans le fichier `.hpp` , exception faite des codes *inline*.

3. Ajouter à la classe `Vandermonde` un constructeur par défaut qui initialise tous les champs privés à 0, et un autre constructeur `Vandermonde(const std::vector<double> &)` qui prend un vecteur en argument et copie ce vecteur dans `coeff` , puis met à jour les autres champs en conséquence.

1. On doit trouver normalement $e_0(1, 10, 100) = 1$, $e_1(1, 10, 100) = 111$, $e_2(1, 10, 100) = 1110$ et $e_3(1, 10, 100) = 1000$.

4. Écrire un accesseur des coefficients de `mat` et un autre de `nb_coeff` .

5. Surcharger l'opérateur d'écriture dans un flux de sortie

```
std::ostream & operator<<(std::ostream &,const Vandermonde &);
```

de sorte qu'il affiche la matrice de Vandermonde sous la forme

```
1 x_0 ... x_0^(N-1)
1 x_1 ... x_1^(N-1)
...
1 x_N-1 ... x_N-1^(N-1)
```

6. Dans un fichier `vandermonde_matrix.cpp` , lire le fichier `sample.dat` et stocker son contenu dans un vecteur `std::vector<double> v` qui servira d'argument d'initialisation pour un objet `Vandermonde V` . Afficher dans un fichier `test_vandermonde.dat` la matrice de Vandermonde obtenue et comparer le résultat avec le fichier `test_vandermonde_reference.dat` . **Les résultats doivent coïncider, il est préférable de revenir aux questions précédentes si ce n'est pas le cas plutôt que de vouloir faire la suite !**

Le déterminant d'une matrice de Vandermonde peut se calculer de la façon suivante :

$$\det V = \prod_{0 \leq i < j \leq N-1} (x_j - x_i), \tag{3}$$

ce qui permet de constater que V est inversible ssi les x_i sont deux à deux distincts. Le cas échéant, les coefficients de la matrice $U = V^{-1} = (u_{ij})$ se calculent à l'aide des polynômes symétriques élémentaires via la formule

$$u_{ij} = \frac{(-1)^{N-1-i} e_{N-1-i}(x_0, ..., x_{j-1}, x_{j+1}, ..., x_N)}{\prod_{k \neq j}(x_j - x_k)}. \tag{4}$$

7. Écrire une méthode `double det()` qui calcule le déterminant d'une matrice de Vandermonde en utilisant la méthode `determinant()` de `Eigen` et une autre `double det2()` qui calcule le déterminant selon la formule (3). Vérifier que les deux calculs coïncident pour la matrice de Vandermonde `V` de la question 6.

8. Écrire une méthode

```
bool is_invertible();
```

qui renvoie `true` si la matrice de Vandermonde est inversible, et `false` sinon. Vérifier que la matrice `V` de la question 6 est bien inversible. Pour une commodité visuelle, il est possible d'afficher les booléens en `true-false` au lieu de `1-0` en écrivant `std::boolalpha` dans le terminal (optionnel).

9. Écrire une méthode

```
Eigen::Matrix<double,Eigen::Dynamic,Eigen::Dynamic> vinverse();
```

qui renvoie la matrice inverse de la matrice de Vandermonde d'après la formule (4).

10. Écrire l'inverse de la matrice `V` dans un fichier `test_vandermonde2.dat` .

11. Écrire un opérateur de multiplication

```
Eigen::Matrix<double,Eigen::Dynamic,Eigen::Dynamic> operator*(const Vandermonde &,
const Eigen::Matrix<double,Eigen::Dynamic,Eigen::Dynamic> &);
```

qui permet de multiplier une matrice de Vandermonde avec une autre matrice de même taille, et vérifier que `V*(V.inverse())` donne bien l'identité (en arrondissant à 0 les coefficients en-dessous d'un certain seuil, par exemple `1e-001`) en affichant le résultat du calcul dans le terminal.

12. Comparer le temps de calcul de l'inverse de `V` en utilisant la méthode `vinverse` et en utilisant la méthode `inverse` de la bibliothèque `Eigen` (qui s'applique comme une méthode classique : si `M` est une matrice, alors on obtient son inverse par `M.inverse()` [2]). On pourra utiliser la bibliothèque `<chrono>` de `C++ 11` [3]. Pour déterminer le temps effectué par un calcul donné, il suffit de procéder comme suit :

```
auto t1 = std::chrono::system_clock::now();//On démarre le chronomètre

...//On effectue le calcul

auto t2 = std::chrono::system_clock::now();//On arrête le chronomètre
std::chrono::duration<double> diff = t2-t1;
std::cout << "Il s'est ecoule " << diff.count() << "s. << std::endl;
```

2. Comme `V.mat` est un champ privé, il faut soit passer par un accesseur de `mat` soit écrire une méthode qui fait appel à `V.mat.inverse()` .

3. Ne pas oublier l'option de compilation idoine !