Travaux dirigés Produit matriciel n°2

Mathématiques pour l'informatique

—IMAC 2—

▸ Exercice 1. *Premiers contacts avec Eigen*

1. *Récupérez la bilbiothèque* `eigen`
   - *sur les machines de la fac, regardez si vous l'avez déjà dans* `/usr/include/eigen3`
   - *sur Linux :* `sudo apt install libeigen3-dev`
   - *sur Mac :* `brew install eigen`
   - *sur windows : téléchargez et installez* `Eigen 3` *(http://eigen.tuxfamily.org)*
2. *Testez l'exemple* `eigenSample.cpp` *sur la page de l'enseignant, en compilant avec :*
   - *pour tous : le* `cmake` *fourni dans le code*

   ou bien si vous préférez compiler vous même :
   - *pour Linux :*
     `g++ -Wall -O2 -I /usr/include/eigen3 file.cpp -o tp2ex1`
   - *pour Mac :*
     `g++ -Wall -O2 -std=c++11 -I /usr/local/include/eigen3 file.cpp -o tp2ex1`
   - *pour Mac M1 :*
     `g++ -Wall -O2 -std=c++11 -I /opt/homebrew/include/eigen3 file.cpp -o tp2ex1`
   - *pour un téléchargement :*
     `g++ -Wall -O2 -I /[path to dir]/eigen3 file.cpp -o tp2ex1`
3. *Regardez la doc et la cheat sheet. Avant de poser vos questions sur* `Eigen`*, jettez y un œil.*

▸ Exercice 2. *Produit scalaire*

1. *Soient x_m un vecteur de taille m et y_n un vecteur de taille n. Quelles conditions sur m et n doivent être satisfaites pour pouvoir effectuer le produit scalaire $x_m \cdot y_n$?*
2. *Codez une fonction produit scalaire en C++ en utilisant les vecteurs de la bibliothèque* Eigen. *Votre fonction aura le prototype suivant :*
   `double dot_product(const Eigen::VectorXd &v1, const Eigen::VectorXd &v2)`
3. *Quelle est la commande pour calculer un produit scalaire avec* `Eigen`*? Comparez le résultat de votre produit scalaire avec celui de* `Eigen`*.*

4. *Pour varier vos test, vous pouvez générer des gros vecteurs contenant des valeurs aléatoires avec les instructions suivantes :*

```
#include <chrono>

[...]

unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
srand(seed);

Eigen::VectorXd x1 = Eigen::VectorXd::Random(vectorSize);
```

5. *Donnez la complexité en temps du produit scalaire.*

6. *Comparez le temps de calcul de votre produit scalaire avec celui de* `Eigen`*. Inspirez-vous du code suivant :*

```
#include <chrono>

[...]

const unsigned int iter = 10000;
auto start = std::chrono::steady_clock::now();
for(unsigned int i=0; i<iter; ++i)
    my_function();
auto end = std::chrono::steady_clock::now();
std::chrono::duration<double> elapsed_seconds = end-start;
std::cout << "elapsed time: " << elapsed_seconds.count() << " s" << std::endl;
```

▸ Exercice 3. *Multiplication de deux matrices*

1. *Soient A_{mk} une matrice à m lignes et k colonnes et B_{ln} une matrice à l lignes et n colonnes. Quelles conditions sur l, m, n et k doivent être satisfaites pour pouvoir effectuer le produit $A_{mk}B_{ln}$?*

2. *Codez en C++ cette multiplication. Votre fonction aura le prototype suivant :*
   `MatrixXd matrix_product(const MatrixXd &m1, const MatrixXd &m2)`

3. *Faites de même en utilisant la fonction du produit scalaire de l'exercice précédent.*

4. *Donnez la complexité en temps de la multiplication de ces matrices. Généraliser pour des matrices carrées $n \times n$.*

5. *Comparez les temps de calcul avec le produit matriciel de* `Eigen`*.*

6. *Si vous voulez estimer la similarité de deux matrices* `A` *et* `B` *(issues de différentes fonctions de produit matricielle par exemple), vous pouvez, pour des matrices de taille relativement grosse, utiliser la fonction suivante :*
   `double error = (A-B).norm() / (A.rows()*A.cols());`
   Expliquez pourquoi cette formule renseigne sur la similitude entre les 2 matrices, sachant que $|\mathrm{A}| = \sqrt{\sum_{i=\mathrm{rows}} \sum_{j=\mathrm{cols}} \mathrm{A}_{ij}^2}$.

▸ **Exercice 4.** ***Matrice de permutation***
Soit $A_{4\times4}$ une matrice carrée d'ordre 4.

1. *Montrer qu'une matrice de permutation est carrée.*
2. *Trouver une matrice M telle que le produit MA renvoie une matrice où les lignes 2 et 4 de A ont été permutées. Vérifier avec Eigen.*
3. *Trouver une matrice N permettant de permuter les colones 2 et 4 de A. Vérifier avec Eigen.*

▸ **Exercice 5.** ***Matrices et C++***
Proposez une structure de données efficace pour traiter les matrices en C++.