Projet C++ : Manipulation de données tensorielles et Décompositions en valeurs singulières

M1 Mathématiques Appliquées 2021-2022

Règles du jeu

- Ce projet est à effectuer et à rendre sous forme d'une archive à transmettre par courriel à l'adresse khitem.blidaoui@dauphine.psl.eu;
- La date limite de rendu du projet est fixée au 3 juin 2022;
- L'évaluation du projet portera sur les éléments suivants :
  - Validation du code via des scripts de tests;
  - Respect des instructions données;
  - Organisation et clarté du code.
- Les consignes de programmation suivantes devront être respectées :
  - Les membres données d'une classe ne pourront pas avoir le statut public;
  - Les différentes classes et fonctions à implémenter ne devront pas copier des librairies ou fonctions existantes et/ou disponibles en ligne;
  - L'usage des librairies standard cmath et iostream est autorisé;
  - Le projet est conçu pour se complexifier au fur et à mesure de l'implémentation. Pour chacune des parties, il est demandé de fournir un script de test validant l'implémentation.

0 Introduction

Un *tenseur d'ordre* d est un tableau de données multi-dimensionnel. Algébriquement, on le notera

$$\mathcal{T} = [\mathcal{T}_{i_1,i_2,\cdots,i_d}] \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}. \tag{0.1}$$

Les entiers $n_1, n_2, \ldots, n_d$ sont appelés les dimensions du tenseur. Ainsi, un vecteur de $\mathbb{R}^{n_1}$ est un tenseur d'ordre 1 de dimension n_1, tandis qu'une matrice de $\mathbb{R}^{n_1 \times n_2}$ est un tenseur d'ordre 2 de dimensions n_1 et n_2.

Les problématiques modernes en analyse de données font de plus en plus intervenir de telles structures tensorielles, qui permettent de modéliser des interactions ou des couplages au sein d'un jeu de données : du point de vue statistique, il peut par exemple s'agir de modéliser des moments d'ordre élevé d'une distribution empirique. Dans le même temps, le traitement de données massives sur des supercalculateurs requiert la réalisation de calculs rapides : là encore, le format tensoriel se révèle adapté à ce contexte [3].

Les problèmes liés au traitement de données tensorielles sont multiples, et s'avèrent significativement plus difficiles dans le cas général que dans le cas de tenseurs d'ordre 1 (vecteurs) ou 2 (matrices). Il reste ainsi beaucoup de chemin à parcourir avant que les librairies de calcul sur les tenseurs soient aussi matures que celles sur les matrices.

Ce projet vise à définir différentes classes de tenseurs ainsi que des opérations associées. On s'intéressera notamment à la décomposition en valeurs singulières, un outil majeur de calcul matriciel utilisé pour la compression et l'approximation de données matricielles, et on en implémentera une généralisation aux tenseurs d'ordre supérieur à 2.[3]

Ce projet est organisé en cinq parties et chacune d'entre elles se base sur les réalisations des parties précédentes.[4] Les parties 1 et 2 proposent un traitement spécifique des tenseurs d'ordres 1 et 2, correspondant respectivement aux vecteurs et aux matrices. La partie 3 consiste en l'implémentation d'un algorithme de calcul de décomposition en valeurs singulières. Dans la partie 4, une structure de données pour les tenseurs d'ordre arbitrairement élevée sera proposée. Enfin, la partie 5 est dédiée à l'implémentation d'une extension de la décomposition en valeurs singulières aux tenseurs quelconques.

0.1 Conventions de notation

Remarque préliminaire importante : *On n'oubliera pas que les indices de tableaux commencent à 0 en C++.*

Dans le reste du sujet, on adoptera les conventions suivantes :

- Pour deux entiers i, j, on notera $i : j$ l'ensemble des entiers $i, i+1, \ldots, j-1, j$ (un tel ensemble pouvant être vide si $i > j$); en particulier, l'instruction `for` $i = j : k$ sera à comprendre comme une boucle sur l'ensemble des entiers compris entre j et k inclus.
- Pour un réel d, signe(d) vaut 1 si $d \geq 0$ et -1 si $d < 0$.

[3] Le but premier de ce projet est la vérification de la maîtrise des concepts liés au C++ : à ce titre, les structures de données que nous nous proposons d'utiliser ne seront pas nécessairement optimales. Elles devraient cependant permettre de traiter des jeux de données de taille modeste.

[4] À une exception près : la partie 3 ne dépend pas de la partie 4.

- Les notations minuscules en gras ($\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$, etc) désigneront des vecteurs, pour les différencier des scalaires (a, b, c, etc). Les composantes d'un vecteur $\boldsymbol{x} \in \mathbb{R}^n$ seront notées $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$: la notation $\boldsymbol{x}_{i:j}$ désignera le vecteur de $\mathbb{R}^{j-i+1}$ dont les composantes seront formées par $\boldsymbol{x}_i, \ldots, \boldsymbol{x}_j$. La transposée d'un vecteur sera notée $\boldsymbol{x}^{\mathrm{T}}$, et on utilisera la notation $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y}$ pour désigner le produit scalaire de deux vecteurs de même dimension. Le premier vecteur de la base canonique de $\mathbb{R}^n$ sera noté $\boldsymbol{e}^1$.

- Les matrices seront identifiées par des lettres majuscules en gras : $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$, etc. Les coefficients d'une matrice $\boldsymbol{A}$ seront identifiés par $\boldsymbol{A}_{i,j}$. La notation $\boldsymbol{A}_{i:j,k:l}$ désignera la sous-matrice de $\boldsymbol{A}$ formées par les lignes i à j et les colonnes k à l de la matrice d'origine. La matrice transposée de $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ sera notée $\boldsymbol{A}^{\mathrm{T}} \in \mathbb{R}^{n\times m}$. La matrice $\boldsymbol{I}[d]$ désignera la matrice identité de l'espace des matrices carrées $\mathbb{R}^{d\times d}$.

- Les tenseurs d'ordre arbitraire seront identifiés par des lettres cursives majuscules : $\mathcal{A}, \mathcal{B}, \mathcal{C}$, etc. Comme dans (0.1), on utilisera un multi-indice $i_1, i_2, \cdots, i_d$ pour identifier les composants du tenseur.

Par abus de langage, on confondra parfois un vecteur/une matrice/un tenseur avec l'objet (l'instance d'une classe C++) censé le représenter.

1 La classe `Vecteur`

1.1 Instructions

Créer une classe `Vecteur` qui représentera un tenseur d'ordre 1 de réels au format `float`. La classe devra comporter les éléments suivants :

- Un membre donnée `tab` de type `float *` qui contiendra les composantes du vecteur représenté;

- Un membre donnée `dim` de type `int`, qui correspondra à la taille du tableau `tab`;

- Une fonction `affiche` sans argument ni valeur de retour qui affiche les valeurs de `dim` ainsi que les éléments de `tab`;

- Un constructeur avec un argument entier qui créera un vecteur représentant un tableau de réels au format `float` de la taille correspondant à l'argument, où les coefficients seront initalisés à 0;

- Un constructeur avec deux arguments, respectivement de type `float *` et `int`, qui créera un vecteur représentant un tableau de réels au format `float` de la taille correspondant au second argument, où les coefficients du tableau seront ceux du tableau passé en premier argument;

- Un destructeur;

- Un constructeur de recopie;

- Une surdéfinition de l'opérateur d'affectation;

- Surdéfinition des opérateurs `+` et `-` pour effectuer la somme et la soustraction de deux vecteurs de même taille;
- Surdéfinition de l'opérateur `[ ]` pour pouvoir accéder aux éléments du tableau de façon à pouvoir les modifier;
- Une fonction `subvec` prenant en argument deux entiers $i \leq j$ et renvoyant un nouvel objet de type `Vecteur` dont le tableau contiendra les composantes $i, i+1, \ldots, j$ du tableau du vecteur appelant.

On définira également les fonctions suivantes :

- La fonction `dot` prendra en arguments deux objets de type `Vecteur` correspondant à deux vecteurs de même taille et renverra leur produit scalaire;
- La fonction `norm` prendra en argument un objet de type `Vecteur` et renverra sa norme euclidienne, ou norme ℓ_2;
- Une surcharge de l'opérateur `*` pour deux arguments de type `float` et `Vecteur`, qui renvoie un objet de type `Vecteur` dont le tableau correspondra à la multiplication du tableau du second argument par le réel en premier argument.

1.2 Validation

Implémenter un programme de tests réalisant les opérations suivantes :

1. Création et affichage des vecteurs $\boldsymbol{u} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \in \mathbb{R}^3$ et $\boldsymbol{v} = \begin{bmatrix} 3 \\ 4 \\ 0 \\ 0 \end{bmatrix} \in \mathbb{R}^4$;
2. Copie de $\boldsymbol{u}$ dans un vecteur $\boldsymbol{t}$;
3. Modification de $\boldsymbol{u}$ en le vecteur $\begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$, et affichage de $\boldsymbol{u}$ et $\boldsymbol{t}$;
4. Calcul du produit scalaire $\boldsymbol{v}^{\mathrm{T}}\boldsymbol{v}$ ainsi que de la norme de $\boldsymbol{v}$;
5. Calcul et affichage du vecteur $\frac{1}{\|\boldsymbol{v}\|}\boldsymbol{v}$;
6. Calcul du vecteur $\boldsymbol{w} = \begin{bmatrix} 4 \\ 0 \\ 0 \end{bmatrix} \in \mathbb{R}^3$ via la fonction `subvec` appliquée à $\boldsymbol{v}$ et affichage des vecteurs $\boldsymbol{v}$ et $\boldsymbol{w}$;
7. Calcul et affichage des vecteurs $\boldsymbol{u} + \boldsymbol{w}$ et $\boldsymbol{u} - \boldsymbol{w}$.

2 La classe `Matrice`

2.1 Instructions

Créer une classe `Matrice` représentant un tenseur d'ordre 2. Cette classe comportera les membres données suivants :

- un tableau `mat` d'objets de type `Vecteur` représentant les colonnes de la matrice;
- un tableau `dims` de deux entiers contenant le nombre de lignes (en position 0) et le nombre de colonnes (en position 1) de la matrice.

La classe comportera également les fonctions membres suivantes :

- Une fonction `affiche` sans argument ni type de retour, qui affiche les dimensions et les coefficients de la matrice;
- Un constructeur prenant en arguments deux entiers correspondant au nombre de lignes et de colonnes de la matrice à représenter, et construisant un tableau rempli de vecteurs nuls;
- Un constructeur prenant en arguments un objet de type `Vecteur` et construisant une matrice carrée diagonale dont les éléments diagonaux seront données par les composantes de ce vecteur;
- Un constructeur prenant en arguments un tableau (de type `Vecteur *`) ainsi que sa taille et construisant une matrice dont les colonnes seront données par les vecteurs du tableau;
- Un destructeur;
- Un constructeur de recopie;
- Une surdéfinition de l'opérateur d'affectation;
- Une surdéfinition de l'opérateur `[ ]` afin de pouvoir accéder à chacun des objets de type `Vecteur` du tableau `mat`;
- Une surdéfinition des opérateurs `+`, `-` et `*` pour coder respectivement la somme, la différence et le produit de deux matrices de tailles compatibles;
- Une fonction `mvprod` prenant en argument un objet de type `Vecteur` dont la dimension correspond au nombre de colonnes de la matrice appelante, et retournant un `Vecteur` correspondant au produit de la matrice appelante avec le vecteur en argument;
- Une fonction `transpose` permettant de construire un nouvel objet de type `Matrice` correspondant à la transposée de la matrice appelante (on rappelle que si $\boldsymbol{B} = \boldsymbol{A}^{\mathrm{T}}$, alors $\boldsymbol{B}_{i,j} = \boldsymbol{A}_{j,i}$ pour tout couple (i, j) indexant un élément de $\boldsymbol{B}$);
- Une fonction `submat` prenant en argument quatre entiers $i_\ell \leq j_\ell$ et $i_c \leq j_c$, et renvoyant un objet de type `Matrice` dont le tableau sera formé par les lignes i_ℓ à j_ℓ et les colonnes i_c à j_c de la matrice appelante.

Enfin, les fonctions ci-dessous seront implémentées :

- Une fonction `norm` renvoyant la norme de Frobenius de la matrice[5];
- Une surdéfinition de l'opérateur `*` prenant en argument un réel de type `float` et un objet de type `Matrice`, qui renvoie un objet de type `Matrice` correspondant à la multiplication de la matrice du second argument par le réel en premier argument;
- Une fonction `outer` prenant en arguments deux objets de type `Vecteur` représentant deux vecteurs de même dimension, et renvoyant un objet de type `Matrice` représentant la matrice défini par le produit externe de ces vecteurs. Si $\boldsymbol{u} \in \mathbb{R}^n$ et $\boldsymbol{v} \in \mathbb{R}^n$, le produit externe est donné par
$$\boldsymbol{u}\boldsymbol{v}^\top = [\boldsymbol{u}_i \boldsymbol{v}_j]_{\substack{1 \leq i \leq n \\ 1 \leq j \leq n}} \in \mathbb{R}^{n \times n}.$$

2.2 Validation

Écrire un programme de tests implémentant les opérations suivantes :

1. Création et affichage des deux matrices ci-dessous :
$$\boldsymbol{A} = \begin{bmatrix} 1 & -1/2 & 0 \\ 1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} -2 & 3 \\ 0 & 1 \end{bmatrix};$$
2. Copie de $\boldsymbol{B}$ dans $\boldsymbol{C}$, changement de valeur $\boldsymbol{B}_{1,2} = 0$ puis affichage de $\boldsymbol{B}$ et $\boldsymbol{C}$;
3. Construction et affichage de $\boldsymbol{D} \in \mathbb{R}^{3 \times 2}$ telle que $\boldsymbol{D} = \boldsymbol{A}_{1:3,1:2}$;
4. Construction et affichage d'une matrice diagonale $\boldsymbol{E}$ dont la diagonale sera donnée par le vecteur $\boldsymbol{v} = \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}$;
5. Calcul et affichage de $\boldsymbol{B} + \boldsymbol{C}$, $\boldsymbol{C} - \boldsymbol{B}$ et $\boldsymbol{D} * \boldsymbol{C}$;
6. Calcul de la norme de Frobenius de $\boldsymbol{C}$;
7. Calcul et affichage de $0.5 * (\boldsymbol{B} + \boldsymbol{B}^{\mathrm{T}})$;

3 Décomposition en valeurs singulières (SVD) pour les matrices

On rappelle que toute matrice $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ admet une *décomposition en valeurs singulières* de la forme :
$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{T}}, \tag{3.1}$$

[5] On rappelle que la norme de Frobenius d'une matrice $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ est donnée par $\|\boldsymbol{A}\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} [\boldsymbol{A}_{ij}]^2}$.

où $\boldsymbol{U} \in \mathbb{R}^{m\times m}$ et $\boldsymbol{V} \in \mathbb{R}^{n\times n}$ sont des matrices orthogonales, et $\boldsymbol{\Sigma} \in \mathbb{R}^{m\times n}$ est telle que $[\boldsymbol{\Sigma}]_{ij} = 0$ si $i \neq j$ et $[\boldsymbol{\Sigma}]_{ii} = \sigma_i \geq 0$. Cette décomposition est d'un intérêt fondamental en analyse de données, et on la désigne généralement par l'acronyme *SVD*[6].

Le but de cette partie est d'implémenter un algorithme de calcul de décomposition en valeurs singulières; l'algorithme choisi est relativement basique, d'autres versions plus élaborées peuvent être trouvées dans l'ouvrage de référence de Golub et Van Loan [2].

L'algorithme de calcul de SVD repose sur plusieurs ingrédients modélisant des transformations de matrice. En premier lieu, les *rotations de Givens* sont des matrices qui permettent des rotations sur deux coordonnées de l'espace (c'est-à-dire deux lignes/deux colonnes d'une matrice) tout en laissant les autres inchangées. Ces matrices ont l'avantage de pouvoir être représentées par deux réels uniquement (le cosinus et le sinus de l'angle de la rotation), ce qui simplifie considérablement leur application (cf Algorithme 3).

Le second ingrédient de l'algorithme est la construction de vecteurs de Householder. Étant donné un vecteur $\boldsymbol{x} \in \mathbb{R}^n$, une *transformation de Householder* est une matrice orthogonale de la forme $\boldsymbol{P} = \boldsymbol{I}[n] - \beta \boldsymbol{v}\boldsymbol{v}^{\mathrm{T}}$ avec $\boldsymbol{v}_1 = 1$, qui projette $\boldsymbol{x}$ sur $\boldsymbol{e}^1$, le premier vecteur de la base canonique de $\mathbb{R}^n$.

Le troisième ingrédient de l'algorithme est la factorisation d'une matrice symétrique via une factorisation dite QR symétrique. Cette méthode s'applique à une matrice carrée : elle construit tout d'abord une décomposition $\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{T}\boldsymbol{Q}^{\top}$, où $\boldsymbol{Q}$ est une matrice orthogonale et $\boldsymbol{T}$ est une matrice tridiagonale symétrique (tous ses éléments sont nuls sauf ceux sur la diagonale et sur les premières sous- et sur-diagonale). Elle factorise ensuite $\boldsymbol{T}$ de sorte à ce que cette matrice devienne diagonale, et modifie $\boldsymbol{Q}$ ce faisant. On obtient au final $\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{T}\boldsymbol{Q}^{\top}$ avec $\boldsymbol{T}$ diagonale.

Enfin, le dernier ingrédient nécessaire à la décomposition en valeurs singulières est la factorisation QR non symétrique avec pivotage. Étant donnée $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ avec $m \geq n$, cette décomposition s'écrit $\boldsymbol{A}\boldsymbol{\Pi} = \boldsymbol{Q}\boldsymbol{R}$, où $\boldsymbol{\Pi} \in \mathbb{R}^{n\times n}$ est une matrice de permutation (qui représente une permutation des colonnes de $\boldsymbol{A}$), $\boldsymbol{Q}$ est orthogonale.

Pour les deux décompositions ci-dessus, l'algorithme classique modifie la matrice $\boldsymbol{A}$ en stockant l'information des facteurs directement dans la matrice[7]. On se servira donc d'une procédure adaptée pour récupérer les facteurs $\boldsymbol{Q}$ dont nous aurons explicitement besoin par la suite.

3.1 Instructions

- En suivant l'algorithme 1, implémenter une fonction `givens` prenant quatre flottants en paramètres et modifiant les valeurs des deux derniers (correspondant à c et s dans l'algorithme 1).
- En se basant sur le schéma de l'algorithme 2 décrit en annexe, implémenter une fonction `householder` prenant en argument un objet de type `Vecteur` ($\boldsymbol{x}$) ainsi qu'un flottant passé par référence (β) et renvoyant un objet de type `Vecteur` ($\boldsymbol{v}$) tout en modifiant la valeur du flottant.
- Implémenter la factorisation QR symétrique décrites par l'algorithme 4 via une fonction `qrsym`. Cette fonction prendra deux objets de type `Matrice` modifiables en arguments,

[6] De l'anglais *Singular Value Decomposition*.

[7] Plus précisément, la matrice $\boldsymbol{A}$ est modifiée de sorte que sa partie supérieure (diagonale et au-dessus) contienne le facteur $\boldsymbol{R} \in \mathbb{R}^{m\times n}$ triangulaire supérieur et que la partie inférieure contienne les coordonnées (sauf la première toujours égale à 1) des vecteurs de Householder utilisés durant la factorisation, ce qui permet d'obtenir $\boldsymbol{Q}$.

représentant la matrice à factoriser et le facteur $\boldsymbol{Q}$ à calculer. La fonction fera appel à une fonction auxiliaire `reductridiag` basée sur l'algorithme 3, qui aura en paramètre un matrice modifiable et en sortie un objet de type `Matrice`.

- Implémenter ensuite la factorisation QR non symétrique avec pivotage (algorithme 5) via une fonctin `qrpivot`. Cette fonction aura deux objets de type `Matrice` (la matrice à factoriser ainsi que son facteur $\boldsymbol{Q}$) en argument, qu'elle devra modifier. Elle renverra également une matrice correspondant aux différents pivots de colonnes effectués par l'algorithme.

- Implémenter enfin l'algorithme de décomposition en valeurs singulières sous la forme d'une fonction `svd`. Celle-ci prendra en arguments un objet de type `Matrice` représentant la matrice d'origine, ainsi qu'un tableau d'objets de type `Matrice` (de taille 3) qui représentera les matrices de la décomposition $\boldsymbol{U}, \boldsymbol{\Sigma}, \boldsymbol{V}$, dans cet ordre.

3.2 Validation

Écrire un script de tests effectuant les opérations suivantes :[8]

- Affichage du résultat de la fonction `givens` appliquée avec $x = 1$ et $z = 2$;
- Affichage du résultat de la fonction `householder` appliquée aux vecteurs suivants :

$$\boldsymbol{x} = \begin{bmatrix} -1 \\ 0 \end{bmatrix} \in \mathbb{R}^2, \quad \boldsymbol{y} = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix} \in \mathbb{R}^2, \quad \boldsymbol{z} = [-4] \in \mathbb{R}^1;$$

- Vérification que la fonction `qrsym` appliquée à $\boldsymbol{M} = \begin{bmatrix} 10 & -6 \\ -6 & 10 \end{bmatrix}$ renvoie

$$\boldsymbol{Q} = \begin{bmatrix} 0.707107 & 0.707107 \\ -0.707107 & 0.707107 \end{bmatrix}.$$

- Vérification que la fonction `qrpivot` appliquée à $\boldsymbol{M} = \begin{bmatrix} 1 & 1/2 & 1/3 \\ 1/2 & 1/3 & 1/4 \\ 1/3 & 1/4 & 1/5 \end{bmatrix}$ renvoie

$$\boldsymbol{R} = \begin{bmatrix} 1.16667 & 0.45 & 0.642857 \\ 0 & 0.105409 & 0.101645 \\ 0 & 0 & 0.0037646 \end{bmatrix}, \quad \boldsymbol{Q} = \begin{bmatrix} 0.857143 & -0.496929 & -0.135526 \\ 0.428571 & 0.542105 & 0.722806 \\ 0.285714 & 0.677631 & -0.677631 \end{bmatrix}, \quad \boldsymbol{\Pi} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

- Affichage de la décomposition en valeurs singulières des matrices suivantes :

$$\boldsymbol{A} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} 2\sqrt{2} & -2\sqrt{2} \\ -\sqrt{2} & -\sqrt{2} \end{bmatrix}, \quad \boldsymbol{C} = \begin{bmatrix} \frac{1}{2} & \frac{3\sqrt{3}}{2} & 0 \\ \frac{\sqrt{3}}{2} & -\frac{3}{2} & 0 \end{bmatrix}.$$

  Les coefficients non diagonaux des matrices $\boldsymbol{\Sigma}$ seront considérés comme nuls si leur valeur absolue est inférieure à 10^{-6} (précision numérique de l'algorithme).

[8] Ces opérations servent à valider le code de façon incrémentale.

4 Tenseurs d'ordre quelconque

On souhaite maintenant représenter des tenseurs d'ordre arbitrairement élevé. Une approche possible consisterait à exprimer un tenseur d'ordre d comme un tableau de tenseurs d'ordre $d-1$, comme nous l'avons fait pour la classe `Matrice` en utilisant la classe `Vecteur` [9]. Cependant, il paraît évident que ce système nécessiterait de construire tous les ordres de tenseur séparément. On considère donc une construction plus globale, qui range tous les éléments du tenseur dans un tableau unidimensionnel.

Afin de mettre à profit les outils disponibles pour les tenseurs d'ordres 1 et 2, on considèrera des versions vectorielles et matricielles d'un tenseur.

Version vectorielle d'un tenseur Soit $\mathcal{T} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ un tenseur d'ordre d. On notera $\mathrm{vec}(\mathcal{T})$ le vecteur de $\mathbb{R}^N$ représentant une "vectorisation" du tenseur. On a ainsi $N = n_1\, n_2\, \cdots\, n_d$ et pour tout d-uplet $(i_1, i_2, \ldots, i_d)$ d'indices, l'élément $\mathcal{T}_{i_1,i_2,\cdots,i_d}$ du tenseur se trouvera dans le vecteur en position

$$\varphi_{n_1,\cdots,n_d}(i_1,\ldots,i_d) \;=\; i_d + n_d(i_{d-1}-1) + n_d\, n_{d-1}(i_{d-2}-1) + \cdots + n_d\, n_{d-1}\cdots n_2(i_1-1). \quad (4.1)$$

Réciproquement, étant donné un indice $i \in \{1,\ldots,N\}$, l'élement i du vecteur $\mathrm{vec}(\mathcal{T})$ correspond à l'élément $(i_1, i_2, \ldots, i_d) = \varphi^{-1}_{n_1,\cdots,n_d}(i)$ du tenseur originel $\mathcal{T}$. Les indices $i_1, \ldots, i_d$ s'obtiennent via la récurrence suivante :

$$\begin{cases} f_d = i, & i_d = f_d \quad [n_d], \\ f_t = \frac{f_{t+1}-i_{t+1}}{n_{t+1}} + 1, & i_t = f_t \quad [n_t], \quad \forall t = 1,\ldots,d-1, \end{cases} \quad (4.2)$$

où $a[b]$ est égal au reste de la division euclidienne de a par b si celui-ci est non nul, et à b s'il est nul.

Modes Pour un tenseur $\mathcal{T} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$, il existe d façons de le déplier en une matrice : ce procédé est appelé dépliage modal, ou *modal unfolding* en anglais.

Pour tout entier k avec $1 \le k \le d$, on peut ainsi définir une matrice $\mathcal{T}^{(k)} \in \mathbb{R}^{n_k \times (N/n_k)}$ (où $N = \prod_{i=1}^{d} n_i$) mettant les éléments du tenseur en deux dimensions. Les coefficients de cette matrice vérifient la relation suivante :

$$\mathcal{T}_{i_1,\cdots,i_{k-1},i_k,i_{k+1},\cdots,i_d} \;=\; \mathcal{T}^{(k)}_{i_k,j_k}, \qquad j_k = \varphi_{n_1,\cdots,n_{k-1},n_{k+1},\cdots,n_d}(i_1,\ldots,i_{k-1},i_{k+1},\ldots,i_d). \quad (4.3)$$

Ainsi, si $\mathcal{T} = [\mathcal{T}_{i,j,k}] \in \mathbb{R}^{3\times4\times2}$, les trois modes de $\mathcal{T}$ sont donnés par :

$$\mathcal{T}^{(1)} = \begin{bmatrix} \mathcal{T}_{1,1,1} & \mathcal{T}_{1,1,2} & \mathcal{T}_{1,2,1} & \mathcal{T}_{1,2,2} & \mathcal{T}_{1,3,1} & \mathcal{T}_{1,3,2} & \mathcal{T}_{1,4,1} & \mathcal{T}_{1,4,2} \\ \mathcal{T}_{2,1,1} & \mathcal{T}_{2,1,2} & \mathcal{T}_{2,2,1} & \mathcal{T}_{2,2,2} & \mathcal{T}_{2,3,1} & \mathcal{T}_{2,3,2} & \mathcal{T}_{2,4,1} & \mathcal{T}_{2,4,2} \\ \mathcal{T}_{3,1,1} & \mathcal{T}_{3,1,2} & \mathcal{T}_{3,2,1} & \mathcal{T}_{3,2,2} & \mathcal{T}_{3,3,1} & \mathcal{T}_{3,3,2} & \mathcal{T}_{3,4,1} & \mathcal{T}_{3,4,2} \end{bmatrix},$$

$$\mathcal{T}^{(2)} = \begin{bmatrix} \mathcal{T}_{1,1,1} & \mathcal{T}_{1,1,2} & \mathcal{T}_{2,1,1} & \mathcal{T}_{2,1,2} & \mathcal{T}_{3,1,1} & \mathcal{T}_{3,1,2} \\ \mathcal{T}_{1,2,1} & \mathcal{T}_{1,2,2} & \mathcal{T}_{2,2,1} & \mathcal{T}_{2,2,2} & \mathcal{T}_{3,2,1} & \mathcal{T}_{3,2,2} \\ \mathcal{T}_{1,3,1} & \mathcal{T}_{1,3,2} & \mathcal{T}_{2,3,1} & \mathcal{T}_{2,3,2} & \mathcal{T}_{3,3,1} & \mathcal{T}_{3,3,2} \\ \mathcal{T}_{1,4,1} & \mathcal{T}_{1,4,2} & \mathcal{T}_{2,4,1} & \mathcal{T}_{2,4,2} & \mathcal{T}_{3,4,1} & \mathcal{T}_{3,4,2} \end{bmatrix},$$

[9] Voire pour la classe `Vecteur`, si l'on considère qu'un réel est un tenseur d'ordre zéro.

et

$$\mathcal{T}^{(3)} = \begin{bmatrix} \mathcal{T}_{1,1,1} & \mathcal{T}_{1,2,1} & \mathcal{T}_{1,3,1} & \mathcal{T}_{1,4,1} & \mathcal{T}_{2,1,1} & \mathcal{T}_{2,2,1} & \mathcal{T}_{2,3,1} & \mathcal{T}_{2,4,1} & \mathcal{T}_{3,1,1} & \mathcal{T}_{3,2,1} & \mathcal{T}_{3,3,1} & \mathcal{T}_{3,4,1} \\ \mathcal{T}_{1,1,2} & \mathcal{T}_{1,2,2} & \mathcal{T}_{1,3,2} & \mathcal{T}_{1,4,2} & \mathcal{T}_{2,1,2} & \mathcal{T}_{2,2,2} & \mathcal{T}_{2,3,2} & \mathcal{T}_{2,4,2} & \mathcal{T}_{3,1,2} & \mathcal{T}_{3,2,2} & \mathcal{T}_{3,3,2} & \mathcal{T}_{3,4,2} \end{bmatrix}.$$

Produit modal Soient $\mathcal{S} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$, un tenseur d'ordre d, k un entier entre 1 et d, et une matrice $\boldsymbol{M} \in \mathbb{R}^{m_k \times n_k}$. Le produit k-modal de $\boldsymbol{M}$ et $\mathcal{S}$, noté $\mathcal{S} \times_k \boldsymbol{M}$, est le tenseur $\mathcal{T} \in \mathbb{R}^{n_1 \times \cdots n_{k-1} \times m_k \times n_{k+1} \times \cdots \times n_d}$ défini par :

$$\mathcal{T}_{i_1,\cdots,i_{k-1},i,i_{k+1},\cdots,i_d} = \sum_{j=1}^{n_k} \boldsymbol{M}_{i,j}\, \mathcal{S}_{i_1,\cdots,i_{k-1},j,i_{k+1},\cdots,i_d}. \tag{4.4}$$

4.1 Instructions

Créer une classe `Tenseur` représentant un tenseur d'ordre arbitraire. Cette classe contiendra les membres données suivants :

- L'ordre d du tenseur (on pourra utiliser un nom de variable différent);
- Un tableau d'entiers de taille d contenant les valeurs des d dimensions du tenseur;
- Un entier `nbelts` correspondant au produit des dimensions du tenseur, et représentant le nombre total d'éléments du tenseur;
- Une version vectorisée du tenseur, c'est-à-dire un objet de type `Vecteur` représentant un tableau de taille `nbelts` contenant les éléments du tenseur dans l'ordre défini en (4.1) par la fonction φ;

La classe `Tenseur` contiendra aussi les fonctions membres listées ci-dessous :

- Un constructeur prenant en arguments un tableau d'entiers ainsi que sa taille, et construisant un tenseur d'ordre et de dimensions correspondantes dont tous les coefficients seront initialisés à 0;
- Un constructeur prenant en arguments un tableau d'entiers, sa taille d, ainsi qu'un objet de type `Vecteur` et construisant un tenseur d'ordre d dont la version vectorisée sera initialisée avec l'objet `Vecteur`;
- Un constructeur prenant en arguments un tableau d'entiers, sa taille d, un entier k entre 1 et d et un objet de classe `Matrice` représentant une matrice $\boldsymbol{A}$: ce constructeur construira un tenseur d'ordre d dont la version vectorisée sera initialisée avec les coefficients de $\boldsymbol{A}$ de sorte que $\boldsymbol{A}$ représente le k-ième mode du tenseur;
- Le destructeur;
- Le constructeur de recopie;
- Surcharge de l'opérateur d'affectation;
- Surcharge de l'opérateur `[ ]` pour qu'il permette d'accéder aux coefficients de la version vectorisée du tenseur;

- Surcharge des opérateurs `+` et `-` pour qu'ils permettent d'additionner ou de soustraire deux tenseurs de mêmes dimensions (et de même ordre);
- Fonction membre `mode` prenant en argument un entier k entre 1 et d et renvoyant un objet de classe `Matrice` représentant le k-ième mode du tenseur appelant.

Enfin, on implémentera une fonction `pmod` prenant en arguments un objet de classe `Tenseur` représentant un tenseur d'ordre d, un objet de classe `Matrice` et un entier k entre 1 et d, et effectuant le produit k-modal du tenseur par cette matrice. Le résultat sera un objet de classe `Tenseur`.

4.2 Validation

Écrire un programme de tests permettant de réaliser les opérations suivantes :

1. Création et affichage d'un tenseur $\mathcal{T} \in \mathbb{R}^{2\times2\times2}$ ayant tous ses coefficients nuls;
2. Création et affichage d'un tenseur $\mathcal{U} \in \mathbb{R}^{2\times2\times2}$ ayant toutes ses composantes égales à 1;
3. Création et affichage des tenseurs $\mathcal{V} = \mathcal{U} + \mathcal{T}$ et $\mathcal{W} = \mathcal{U} - \mathcal{T}$;
4. Affichage de $\mathcal{U}_{2,2,2}$, modification de $\mathcal{U}_{2,2,2}$ en -1, puis affichage des tenseurs $\mathcal{U}$ et $\mathcal{V}$;
5. Calcul et affichage du mode 1 du tenseur $\mathcal{T}$;
6. Modification des coefficients du tenseur de sorte que le mode 2 du tenseur $\mathcal{T}$ corresponde à la matrice :
$$\begin{bmatrix} 1 & 3 & 0 & -1 \\ 4 & 1/3 & 1.5 & 2 \end{bmatrix}$$
7. Calcul et affichage du tenseur $\mathcal{S} = \mathcal{T} \times_3 \boldsymbol{A}$, où
$$\boldsymbol{A} = \begin{bmatrix} 3 & -1 \\ 0 & 6 \\ 0 & -3 \end{bmatrix};$$
8. Calcul et affichage du tenseur $\mathcal{R} = \mathcal{S} + \mathcal{S}$.

5 Décomposition en valeurs singulières d'ordre élevé

La décomposition en valeurs singulières d'ordre élévé, ou *HOSVD*[10], d'un tenseur d'ordre arbitraire se définit via la SVD de ses différents modes.

Soit $\mathcal{T} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ un tenseur d'ordre d. Pour tout k entre 1 et d, le mode $\mathcal{T}^{(k)}$ admet une décomposition en valeurs singulières que l'on note :

$$\mathcal{T}^{(k)} \;=\; \boldsymbol{U}^{(k)} \boldsymbol{\Sigma}^{(k)} \left[\boldsymbol{V}^{(k)}\right]^{\mathrm{T}}.$$

On définit alors la décomposition HOSVD de $\mathcal{T}$ [1] comme

$$\mathcal{T} \;=\; \mathcal{S} \times_1 \boldsymbol{U}^{(1)} \times_2 \boldsymbol{U}^{(2)} \cdots \times_d \boldsymbol{U}^{(d)}, \tag{5.1}$$

où

$$\mathcal{S} \;=\; \mathcal{T} \times_1 \left[\boldsymbol{U}^{(1)}\right]^{\mathrm{T}} \times_2 \left[\boldsymbol{U}^{(2)}\right]^{\mathrm{T}} \cdots \times_d \left[\boldsymbol{U}^{(d)}\right]^{\mathrm{T}}$$

est appelé le coeur de tenseur[11]. Les matrices $\boldsymbol{U}^{(1)}, \boldsymbol{U}^{(2)}, \ldots, \boldsymbol{U}^{(d)}$ sont quant à elles appelées les facteurs de la décomposition. [12]

5.1 Instructions

Créer une classe `TenseurSVD` qui dérive de la classe `Tenseur`. Les membres données hérités de la classe `Tenseur` représenteront le coeur de tenseur dans la décomposition. La classe `TenseurSVD` possèdera en plus un tableau de `Matrice` de taille d représentant les différents facteurs de la décomposition en valeurs singulières d'ordre élevé. On réalisera également les implémentations suivantes :

- Les constructeurs de la classe Tenseur seront étendus pour incorporer en argument un tableau de d objets de type `Matrice`.
- Le reste de la forme canonique (destructeur, constructeur de recopie, opérateur d'affectation) sera également redéfini;
- Une fonction membre `tenseurtotal` sera également implémentée : celle-ci renverra un objet de la classe `Tenseur` contenant le produit du coeur de tenseur avec les différents facteurs (c'est-à-dire le tenseur en forme non factorisée).

Une fois la classe `TenseurSVD` définie, écrire une fonction `hosvd` qui implémente la décomposition en valeurs singulières d'ordre élevé. Cette fonction prendra en argument un objet de type `Tenseur` et retournera un objet de type `TenseurSVD`.

---

[10] Pour *High-Order Singular Value Decomposition*.

[11] Le coeur de tenseur est du même ordre que le tenseur originel, par contre les valeurs de ses dimensions peuvent être différentes. En ce sens, ce tenseur synthétise l'information du tenseur de départ.

[12] On remarquera que dans le cas d'une matrice, on a $\boldsymbol{U}^{(2)} = \left[\boldsymbol{V}^{(1)}\right]^{\mathrm{T}}$, ce qui est bien cohérent avec la définition de la SVD.

5.2 Validation

Écrire un script de tests permettant de réaliser la HOSVD du tenseur $\mathcal{T} \in \mathbb{R}^{3\times3\times3}$ dont le mode 1 est donné par

$$\mathcal{T}^{(1)} = \begin{bmatrix} 0.9073 & 0.7158 & -0.3698 & 1.7842 & 1.6970 & 0.0151 & 2.1236 & -0.0740 & 1.4429 \\ 0.8924 & -0.4898 & 2.4288 & 1.7753 & -1.5077 & 4.0337 & -0.6631 & 1.9103 & -1.7495 \\ 2.1488 & 0.3054 & 2.3753 & 4.2495 & 0.3207 & 4.7146 & 1.8260 & 2.1335 & -0.2716 \end{bmatrix}.$$

Ce script créera le tenseur originel via un objet de la classe `Tenseur` (dont on affichera le contenu), puis appellera la fonction `hosvd` pour créer l'objet de type `TenseurSVD` correspondant. Les facteurs et le coeur de tenseur seront affichés séparément, puis la fonction `tenseurtotal` sera appelée, et son résultat sera affiché pour comparaison avec le tenseur d'origine.

L'erreur relative entre les coefficients des deux tenseurs sera affichée. Pour un mode k donné, l'erreur entre deux tenseurs $\mathcal{T}$ et $\tilde{\mathcal{T}}$ sera donné par les coefficients de la matrice

$$\boldsymbol{M} = \frac{1}{\|\mathcal{T}^{(k)}\|_F}\left(\mathcal{T}^{(k)} - \tilde{\mathcal{T}}^{(k)}\right).$$

L'objectif sera d'avoir une erreur de l'ordre de 10^{-3} au plus[13].

Bibliographie

[1] L. De Lathauwer, B. De Moor, and J. Vandewalle. A multilinear singular value decomposition. *SIAM J. Matrix Anal. Appl.*, 21:1253–1278, 2000.

[2] G. H. Golub and C. F. Van Loan. *Matrix computations*. The Johns Hopkins University Press, Baltimore, fourth edition, 2013.

[3] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM Rev.*, 51:455–500, 2009.

[13] Il s'agit là d'une erreur importante, mais dont l'amélioration demandera des outils d'algèbre linéaire numérique allant au-delà de ce cours.

Annexe : algorithmes d'algèbre linéaire

Remarque importante : *Qu'il s'agisse de vecteurs, de matrices ou de tenseurs, on fera toujours commencer les indices à* 1*. Cependant, en C++, les indices commencent à* 0 *: il faudra donc effectuer un décalage d'indices.*

Algorithm 1: Matrice de rotation de Givens

Entrées : Réels x, z.
Sorties : Coefficients de rotation de Givens c, s.

if $z = 0$ **then**
  $c \leftarrow 1,\ s \leftarrow 0$
else
  if $|z| > |x|$ **then**
    $\tau \leftarrow -x/z,\ s \leftarrow 1/\sqrt{1+\tau^2}, c \leftarrow s\tau$
  else
    $\tau \leftarrow -z/x,\ c \leftarrow 1/\sqrt{1+\tau^2}, s \leftarrow c\tau$
  end
end

Algorithm 2: Calcul de matrice de Householder

Entrées : Vecteur $\boldsymbol{x} \in \mathbb{R}^n$.
Sorties : Vecteur $\boldsymbol{v} \in \mathbb{R}^n$, scalaire $\beta \in \mathbb{R}$ avec $\boldsymbol{v}_1 = 1$, $\boldsymbol{P} = \boldsymbol{I}[n] - \beta \boldsymbol{v}\boldsymbol{v}^{\mathrm{T}}$ orthogonale et $\boldsymbol{P}\boldsymbol{x} = \|\boldsymbol{x}\|_2 \boldsymbol{e}^1$.

$\sigma \leftarrow \boldsymbol{x}_{2:n}^{\mathrm{T}} \boldsymbol{x}_{2:n}$.
$\boldsymbol{v} \leftarrow \begin{bmatrix} 1 \\ \boldsymbol{x}_{2:n} \end{bmatrix}$.
if *(*$\sigma = 0$ & $\boldsymbol{x}_1 \geq 0$*)* **then**
  $\beta \leftarrow 0$.
else if *(*$\sigma = 0$ & $\boldsymbol{x}_1 < 0$*)* **then**
  $\beta \leftarrow 2$.
else
  $\mu \leftarrow \sqrt{\boldsymbol{x}_1^2 + \sigma}$.
  if $\boldsymbol{x}_1 \leq 0$ **then**
    $\boldsymbol{v}_1 \leftarrow \boldsymbol{x}_1 - \mu$.
  else
    $\boldsymbol{v}_1 \leftarrow -\frac{\sigma}{\boldsymbol{x}_1 + \mu}$.
  end
  $\beta \leftarrow \frac{2\boldsymbol{v}_1^2}{\sigma + \boldsymbol{v}_1^2}$.
  $\boldsymbol{v} \leftarrow \frac{\boldsymbol{v}}{\boldsymbol{v}_1}$.
end

Algorithm 3: Réduction d'une matrice tridiagonale

Entrées : Matrice tridiagonale $\boldsymbol{D} \in \mathbb{R}^{\ell \times \ell}$
Sorties : Modification de la matrice en $\boldsymbol{Z}^\top \boldsymbol{D} \boldsymbol{Z}$; matrice $\boldsymbol{Z}$

$d \leftarrow (\boldsymbol{D}_{\ell-1,\ell-1} - \boldsymbol{D}_{\ell,\ell})/2$
$\mu \leftarrow \boldsymbol{D}_{\ell,\ell} - \boldsymbol{D}^2_{\ell,\ell-1} / \left(d + \text{signe}(d)\sqrt{d^2 + \boldsymbol{D}^2_{\ell,\ell-1}}\right)$
$x \leftarrow \boldsymbol{D}_{1,1} - \mu$
$z \leftarrow \boldsymbol{D}_{2,1}$
$\boldsymbol{Z} \leftarrow \boldsymbol{I}[\ell]$
for $k = 1, \ldots, \ell - 1$ **do**
    Calculer c et s en appelant l'algorithme 1 avec x et z en paramètres
    `// Application de la rotation de Givens à droite`
    for $j = 1, \ldots, \ell$ **do**
        $\tau_1 \leftarrow \boldsymbol{D}_{j,k}$
        $\tau_2 \leftarrow \boldsymbol{D}_{j,k+1}$
        $\boldsymbol{D}_{j,k} \leftarrow c\tau_1 - s\tau_2$
        $\boldsymbol{D}_{j,k+1} \leftarrow s\tau_1 + c\tau_2$
        $\tau_1 \leftarrow \boldsymbol{Z}_{j,k}$
        $\tau_2 \leftarrow \boldsymbol{Z}_{j,k+1}$
        $\boldsymbol{Z}_{j,k} \leftarrow c\tau_1 - s\tau_2$
        $\boldsymbol{Z}_{j,k+1} \leftarrow s\tau_1 + c\tau_2$
    end
    `// Application de la rotation de Givens à gauche`
    for $j = 1, \ldots, \ell$ **do**
        $\tau_1 \leftarrow \boldsymbol{D}_{k,j}$
        $\tau_2 \leftarrow \boldsymbol{D}_{k+1,j}$
        $\boldsymbol{D}_{k,j} \leftarrow c\tau_1 - s\tau_2$
        $\boldsymbol{D}_{k+1,j} \leftarrow s\tau_1 + c\tau_2$
    end
    if $k < \ell - 1$ **then**
        $x \leftarrow \boldsymbol{D}_{k+1,k}$
        $z \leftarrow \boldsymbol{D}_{k+2,k}$
    end
end

Algorithm 4: Calcul de factorisation QR pour matrice symétrique

Entrées : Matrice $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ symétrique modifiable.
Sorties : Matrice $\boldsymbol{Q} \in \mathbb{R}^{n\times n}$ orthogonale telle que $\boldsymbol{Q}^\top \boldsymbol{A}\boldsymbol{Q}$ soit diagonale.

$\boldsymbol{Q} \leftarrow \boldsymbol{I}[n]$ (identité de $\mathbb{R}^{n\times n}$)
`// Phase 1 : Tridiagonalisation de la matrice :` $\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{T}\boldsymbol{Q}^{\mathrm{T}}$
for $k = 1 : (n-2)$ **do**
  Calculer $\boldsymbol{v}, \beta$ en appelant l'algorithme 2 avec $\boldsymbol{A}_{(k+1):n,k}$ en argument.
  $\boldsymbol{p} \leftarrow \beta \boldsymbol{A}_{(k+1):n,(k+1):n}\, \boldsymbol{v}$
  $\boldsymbol{w} \leftarrow \boldsymbol{p} - (\beta/2) * \left(\boldsymbol{p}^{\mathrm{T}}\boldsymbol{v}\right) \boldsymbol{v}$
  $\boldsymbol{A}_{k+1,k} = \left\|\boldsymbol{A}_{(k+1):n,k}\right\|$
  $\boldsymbol{A}_{k,k+1} = \boldsymbol{A}_{k+1,k}$
  $\boldsymbol{A}_{(k+1):n,(k+1):n} = \boldsymbol{A}_{(k+1):n,(k+1):n} - \boldsymbol{v}\boldsymbol{w}^{\mathrm{T}} - \boldsymbol{w}\boldsymbol{v}^{\mathrm{T}}$.
  $\boldsymbol{Q}_{(k+1):n,(k+1):n} \leftarrow \boldsymbol{Q}_{(k+1):n,(k+1):n} - \beta \boldsymbol{v}\boldsymbol{v}^{\mathrm{T}} \boldsymbol{Q}_{(k+1):n,(k+1):n}$
end
$\boldsymbol{T} \leftarrow \boldsymbol{0}[n]$ (matrice zéro de $\mathbb{R}^{n\times n}$)
for $j = n, n-1, \ldots, 1$ **do**
  $\boldsymbol{T}_{j,j} \leftarrow \boldsymbol{A}_{j,j}$
  if $j > 1$ **then** $\boldsymbol{T}_{j-1,j} \leftarrow \boldsymbol{A}_{j,j-1},\ \boldsymbol{T}_{j,j-1} \leftarrow \boldsymbol{T}_{j-1,j}$
end
`// Phase 2 : Diagonalisation de` $\boldsymbol{T}$ `et mise à jour de` $\boldsymbol{Q}$
while $\boldsymbol{T}$ *n'est pas diagonale* **do**
  for $i = 1, \ldots, n-1$ **do**
    `// Gestion des erreurs numériques`
    if $|\boldsymbol{T}_{i,i+1}| + |\boldsymbol{T}_{i+1,i}| \leq 10^{-9}\,(|\boldsymbol{T}_{i,i}| + |\boldsymbol{T}_{i+1,i+1}|)$ **then**
      $\boldsymbol{T}_{i,i+1} \leftarrow 0, \quad \boldsymbol{T}_{i+1,i} \leftarrow 0.$
    end
  end
  Parcourir $\boldsymbol{T}$ pour trouver deux entiers p et q entre 0 et n tels que $\boldsymbol{T}$ s'écrive

$$\boldsymbol{T} = \begin{bmatrix} \boldsymbol{T}_1 & 0 & 0 \\ 0 & \boldsymbol{T}_2 & 0 \\ 0 & 0 & \boldsymbol{T}_3 \end{bmatrix}$$

  avec $\boldsymbol{T}_1 \in \mathbb{R}^{p\times p}$ et $\boldsymbol{T}_3 \in \mathbb{R}^{q\times q}$ diagonales, et $\boldsymbol{T}_2 \in \mathbb{R}^{(n-p-q)\times(n-p-q)}$ une matrice tridiagonale non diagonale (on parle de matrice non réduite).
  if $p + q < n$ **then**
    Appliquer l'algorithme 3 à $\boldsymbol{T}_2$ de sorte à obtenir $\boldsymbol{Z}$. *(NB : $\boldsymbol{T}_2$ est modifiée en $\boldsymbol{Z}^\top \boldsymbol{T}_2 \boldsymbol{Z}$ par l'algorithme 3.)*
    Mettre à jour $\boldsymbol{T}$ et $\boldsymbol{Q}$ comme suit :

$$\boldsymbol{T} \leftarrow \frac{\hat{\boldsymbol{T}} + \hat{\boldsymbol{T}}^\top}{2} \text{ avec } \quad \hat{\boldsymbol{T}} = \begin{bmatrix} \boldsymbol{T}_1 & 0 & 0 \\ 0 & \boldsymbol{T}_2 & 0 \\ 0 & 0 & \boldsymbol{T}_3 \end{bmatrix}, \qquad \boldsymbol{Q} \leftarrow \boldsymbol{Q} * \begin{bmatrix} \boldsymbol{I}[p] & 0 & 0 \\ 0 & \boldsymbol{Z} & 0 \\ 0 & 0 & \boldsymbol{I}[q] \end{bmatrix}$$

  end
end

Algorithm 5: Calcul de factorisation QR avec pivot

Entrées : Matrice $\boldsymbol{A} \in \mathbb{R}^{m\times n}, m \geq n$ modifiable.
Sorties : Matrice $\boldsymbol{Q} \in \mathbb{R}^{m\times m}$ orthogonale, matrice $\boldsymbol{\Pi}$ de permutation.

$\boldsymbol{\Pi} \leftarrow \boldsymbol{I}[n]$ (matrice identité de $\mathbb{R}^{n\times n}$)
for $j = 1 : n$ **do**
    $\boldsymbol{c}_j \leftarrow \boldsymbol{A}_{1:m,j}^{\mathrm{T}} \boldsymbol{A}_{1:m,j}$
end
$r \leftarrow 0$
$\tau \leftarrow \max\{\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n\}$
while *($\tau > 0$ & $r < n$)* **do**
    $r \leftarrow r + 1$
    Définir k comme le plus petit entier tel que $r \leq k \leq n$ et $\boldsymbol{c}_k = \tau$.
    Permuter $\boldsymbol{A}_{1:m,r}$ et $\boldsymbol{A}_{1:m,k}$
    Permuter $\boldsymbol{c}_r$ et $\boldsymbol{c}_k$
    Permuter $\boldsymbol{\Pi}_{1:n,r}$ et $\boldsymbol{\Pi}_{1:n,k}$
    Calculer $\boldsymbol{v}, \beta$ en appelant l'algorithme 2 avec $\boldsymbol{A}_{r:m,r}$ en argument.
    $\boldsymbol{A}_{r:m,r:n} \leftarrow \boldsymbol{A}_{r:m,r:n} - \beta \boldsymbol{v}\boldsymbol{v}^{\mathrm{T}} \boldsymbol{A}_{r:m,r:n}$
    $\boldsymbol{A}_{(r+1):m,r} \leftarrow \boldsymbol{v}_{2:(m-r+1)}$
    for $i = (r + 1) : n$ **do**
        $\boldsymbol{c}_i \leftarrow \boldsymbol{c}_i - \boldsymbol{A}_{r,i}^2$
    end
    if $r < n$ **then**
        $\tau \leftarrow \max\{\boldsymbol{c}_{r+1}, \ldots, \boldsymbol{c}_n\}$
    else
        $\tau \leftarrow 0$
    end
end
`// Calcul de` $\boldsymbol{Q}$
$\boldsymbol{Q} \leftarrow \boldsymbol{I}[m]$ (identité de $\mathbb{R}^{m\times m}$)
$\boldsymbol{v} \leftarrow \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^m$
for $j = n, n-1, \ldots, 1$ **do**
    $\boldsymbol{v}_{j:m} \leftarrow \begin{bmatrix} 1 \\ \boldsymbol{A}_{(j+1):m,j} \end{bmatrix}$
    $\beta \leftarrow \frac{2}{1+\|\boldsymbol{A}_{(j+1):m,j}\|^2}$
    $\boldsymbol{Q}_{j:m,j:m} \leftarrow \boldsymbol{Q}_{j:m,j:m} - \beta \boldsymbol{v}_{j:m} \boldsymbol{v}_{j:m}^{\mathrm{T}} \boldsymbol{Q}_{j:m,j:m}$
end

Algorithm 6: Calcul de SVD

Entrées : Matrice $\boldsymbol{A} \in \mathbb{R}^{m \times n}$.
Sorties : Factorisation de la matrice $\boldsymbol{A}$ en $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{T}}$.

if $m \geq n$ **then**
    Appliquer l'algorithme 4 à $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ pour obtenir $\boldsymbol{Q}_1 \in \mathbb{R}^{n \times n}$ comme "facteur Q" de la décomposition.
    Appliquer l'algorithme 5 à $\boldsymbol{A}\boldsymbol{Q}_1$ afin d'obtenir $\boldsymbol{Q}_2 \in \mathbb{R}^{m \times m}$, $\boldsymbol{\Pi}$.
    Calculer $\boldsymbol{R} = \boldsymbol{Q}_2^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{Q}_1)\boldsymbol{\Pi}$.
    for $j = 1, \ldots, n$ **do**
        if $\boldsymbol{R}_{jj} < 0$ **then**
            $[\boldsymbol{Q}_2]_{1:m,j} \leftarrow -[\boldsymbol{Q}_2]_{1:m,j}$
        end
    end
    Recalculer $\boldsymbol{R} = \boldsymbol{Q}_2^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{Q}_1)\boldsymbol{\Pi}$.
    Renvoyer $\boldsymbol{U} = \boldsymbol{Q}_2, \boldsymbol{\Sigma} = \boldsymbol{R}$ et $\boldsymbol{V} = \boldsymbol{Q}_1\,\boldsymbol{\Pi}$.
else
    Appliquer l'algorithme 4 à $\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}}$ pour obtenir $\boldsymbol{Q}_1 \in \mathbb{R}^{m \times m}$ comme facteur Q de la décomposition.
    Appliquer l'algorithme 5 à $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{Q}_1$ afin d'obtenir $\boldsymbol{Q}_2 \in \mathbb{R}^{n \times n}$ et $\boldsymbol{\Pi}$.
    Calculer $\boldsymbol{R} = \boldsymbol{Q}_2^{\mathrm{T}}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{Q}_1)\boldsymbol{\Pi}$.
    for $i = 1, \ldots, m$ **do**
        if $\boldsymbol{R}_{ii} < 0$ **then**
            $[\boldsymbol{Q}_2]_{1:n,i} \leftarrow -[\boldsymbol{Q}_2]_{1:n,i}$
        end
    end
    Recalculer $\boldsymbol{R} = \boldsymbol{Q}_2^{\mathrm{T}}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{Q}_1)\boldsymbol{\Pi}$.
    Renvoyer $\boldsymbol{U} = \boldsymbol{Q}_1\,\boldsymbol{\Pi}, \boldsymbol{\Sigma} = \boldsymbol{R}^{\mathrm{T}}$ et $\boldsymbol{V} = \boldsymbol{Q}_2$.
end