Détails du projet C++

Clément Royer

M1 Mathématiques et Applications - Parcours Mathématiques Appliquées

Version du 6 juin 2020

Modalités du projet

- Date de rendu : entre les 5 et 7 juin 2020.
- Envoi par mail à l'adresse :
  `clement.royer@dauphine.psl.eu`

Attendus

- Ensemble des fichiers .h et .cpp nécessaires au fonctionnement du code;
- **Conseillé :** fichier README.

Outline

1. Projet - Point de vue théorique
2. Projet - Point de vue implémentation
3. Questions/Réponses

Sommaire

1. Projet - Point de vue théorique
2. Projet - Point de vue implémentation
3. Questions/Réponses

Sujet du projet (1/2) : SVD

Décomposition en valeurs singulières (SVD) [Eckhart, Young 1936]

Toute matrice $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ s'écrit $\boldsymbol{A} = \boldsymbol{U\Sigma V}^\top$ avec :

- $\boldsymbol{U} \in \mathbb{R}^{m \times m}$ orthogonale ($\boldsymbol{U}^{-1} = \boldsymbol{U}^\top$);
- $\boldsymbol{\Sigma} \in \mathbb{R}^{m \times n}$ avec

$$\boldsymbol{\Sigma}_{ij} = \begin{cases} \sigma_i \geq 0 & \text{si } i = j \\ 0 & \text{sinon.} \end{cases}$$

- $\boldsymbol{V} \in \mathbb{R}^{n \times n}$ orthogonale.

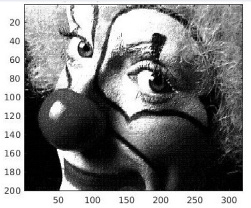
Intérêt de la SVD

- **Décomposition** : Meilleure représentation de l'information;
- **Approximation** : Garder les plus grandes valeurs singulières est la meilleure approximation possible
  $\Rightarrow$ Idée derrière l'analyse en composantes principales, la projection, etc.

Application de la SVD

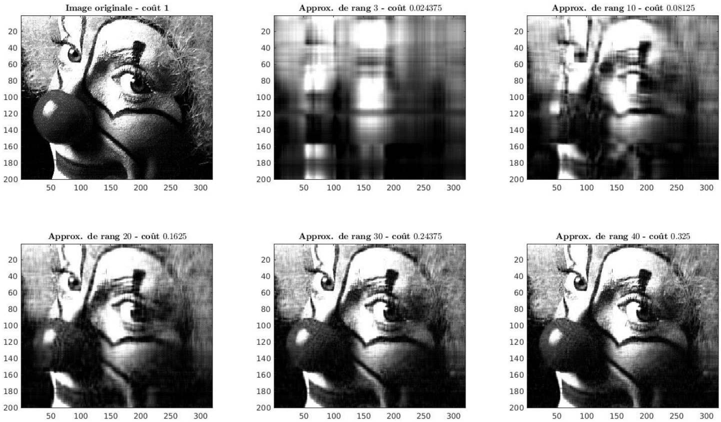
Compression d'images

On considère une image 200x320 pixels stockée sous la forme d'une matrice $\boldsymbol{A} \in \mathbb{R}^{m \times n}$.



- On calcule la *SVD* de $\boldsymbol{A} \Rightarrow \boldsymbol{U}, \boldsymbol{\Sigma}, \boldsymbol{V}$;
- $\boldsymbol{A}$ est de rang 200.
- On teste plusieurs *SVD* tronquées : $\boldsymbol{A}_k = \boldsymbol{U}_k \boldsymbol{\Sigma}_{k,k} \boldsymbol{V}^{\mathrm{T}}$ pour différentes valeurs de k.

Application de la SVD (2)



Sujet du projet (2/2) : les tenseurs

Les tenseurs

- Structures de données multidimensionnelles;
- Généralisent les vecteurs et matrices;
- Utilisées dans les GPUs, ainsi qu'en analyse de données.

Extension de la SVD aux tenseurs [DeLathauwer et al, 2000]

- Un tenseur d'ordre d peut être représenté de d façons différentes par une matrice (modes);
- On amalgame les SVD de chacune de ces matrices pour former la décomposition HOSVD.

Organisation du projet

Cinq parties

1. Vecteurs
2. Matrices
3. SVD pour les matrices;
4. Tenseurs;
5. HOSVD.

- Chaque partie se base sur les précédentes;
- La manipulation des tenseurs se fait via des vecteurs et des matrices;
- La partie 3 requiert plusieurs algorithmes d'algèbre linéaire numérique.

Partie 1 du projet : les vecteurs

En maths : tableau à une entrée

$$\boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{x} = [\boldsymbol{x}_i]_{i=1,\dots,n} = \begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_n \end{bmatrix}.$$

Pour le projet C++

- Classe `Vecteur` contenant un tableau de réels flottants et sa dimension $\Rightarrow$ Allocation dynamique
- Plusieurs constructeurs + Forme canonique;
- Opérateurs à surdéfinir : `+`, `-`, `[]`
- Norme et produit scalaire :

$$\boldsymbol{x}^\top \boldsymbol{y} = \sum_{i=1}^{n} \boldsymbol{x}_i \boldsymbol{y}_i, \quad \|\boldsymbol{x}\| = \sqrt{\boldsymbol{x}^\top \boldsymbol{x}}.$$

Partie 2 du projet : les matrices

En maths : Tableau à deux entrées

$$\boldsymbol{A} \in \mathbb{R}^{m \times n}, \quad \boldsymbol{A} = [\boldsymbol{A}_{ij}]_{\substack{1 \le i \le m \\ 1 \le j \le n}} = \begin{bmatrix} \boldsymbol{A}_{1,1} & \cdots & \boldsymbol{A}_{1,n} \\ \vdots & \ddots & \vdots \\ \boldsymbol{A}_{m,1} & \cdots & \boldsymbol{A}_{m,n} \end{bmatrix}.$$

Pour le projet C++

- Un tableau d'objets de type `Vecteur` (colonnes de la matrice);
- Les dimensions de la matrice;
- Constructeurs, forme canonique;
- Surdéfinitions : `+`,`-`,`[ ]`,`*`
  (*Produit matriciel :* Si $\boldsymbol{C} = \boldsymbol{A} * \boldsymbol{B}$ avec $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ et $\boldsymbol{B} \in \mathbb{R}^{n \times p}$, alors $\forall i = 1..m, \forall j = 1..p,\ \boldsymbol{C}_{i,j} = \sum_{k=1}^{n} \boldsymbol{A}_{i,k} \boldsymbol{B}_{k,j}$.)
- *Norme de Frobenius :* $\|\boldsymbol{A}\|_F^2 = \sum_{i,j} \boldsymbol{A}_{ij}^2$.
- *Transposée :* $\left[\boldsymbol{A}^\top\right]_{ji} = \boldsymbol{A}_{ij}$.

Partie 3 : SVD

L'algorithme

- Version simplifiée de la méthode de référence;
- Décomposé en plusieurs routines standards, cf [Golub and Van Loan, 2013];
- Outil-clé : décompositions matricielles.

Comment naviguer dans ces algorithmes

- Les implémenter un par un;
- Les tester sur les problèmes demandés.

Théorie de la SVD (1/3)

Théorème

Toute matrice $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ symétrique admet une décomposition dite **spectrale** de la forme :

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{-1},$$

avec

- $\boldsymbol{Q}$ matrice orthogonale ($\boldsymbol{Q}^{\mathrm{T}} = \boldsymbol{Q}^{-1}, \det(\boldsymbol{Q}) = 1$), dont les colonnes $\boldsymbol{p}_1, \dots, \boldsymbol{p}_n$ forment une *base orthonormée* de vecteurs propres.
- $\boldsymbol{\Lambda}$ matrice diagonale, qui contient les n valeurs propres de $\boldsymbol{A}$ $\lambda_1, \dots, \lambda_n$ sur la diagonale.

Théorie de la SVD (1/3)

Théorème

Toute matrice $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ symétrique admet une décomposition dite **spectrale** de la forme :

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{-1},$$

avec

- $\boldsymbol{Q}$ matrice orthogonale ($\boldsymbol{Q}^{\mathrm{T}} = \boldsymbol{Q}^{-1}, \det(\boldsymbol{Q}) = 1$), dont les colonnes $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n$ forment une *base orthonormée* de vecteurs propres.
- $\boldsymbol{\Lambda}$ matrice diagonale, qui contient les n valeurs propres de $\boldsymbol{A}$ $\lambda_1, \ldots, \lambda_n$ sur la diagonale.

- Il n'y a pas unicité de la décomposition spectrale;
- Aux permutations près, l'ensemble des valeurs propres est unique.

Théorie de la SVD (2/3)

Matrices rectangulaires

Soit $\boldsymbol{A} \in \mathbb{R}^{m \times n}$. On peut parler :

- des valeurs propres de $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} \in \mathbb{R}^{n \times n}$;
- des valeurs propres de $\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}} \in \mathbb{R}^{m \times m}$.

On peut s'en servir pour obtenir une décomposition de $\boldsymbol{A}$.

Théorie de la SVD (2/3)

Matrices rectangulaires

Soit $\mathbf{A} \in \mathbb{R}^{m \times n}$. On peut parler :

- des valeurs propres de $\mathbf{A}^\mathrm{T}\mathbf{A} \in \mathbb{R}^{n \times n}$;
- des valeurs propres de $\mathbf{A}\mathbf{A}^\mathrm{T} \in \mathbb{R}^{m \times m}$.

On peut s'en servir pour obtenir une décomposition de $\mathbf{A}$.

Observations concernant $\mathbf{A}^\mathrm{T}\mathbf{A}$

- $\mathbf{A}^\mathrm{T}\mathbf{A}$ est symétrique réelle, donc diagonalisable;
- $\mathbf{A}^\mathrm{T}\mathbf{A}$ est semi-définie positive donc ses valeurs propres sont **positives ou nulles**.
- $\mathrm{rang}(\mathbf{A}^T\mathbf{A}) = \mathrm{rang}(\mathbf{A})$.

(On a des résultats similaires pour $\mathbf{A}\mathbf{A}^\mathrm{T}$.)

Théorie de la SVD (3/3)

Théorème

Toute matrice $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ admet une **décomposition en valeurs singulières** (*SVD*) de la forme

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{T}},$$

où

- $\boldsymbol{U} \in \mathbb{R}^{m \times m}$ est orthogonale;
- $\boldsymbol{V} \in \mathbb{R}^{n \times n}$ est orthogonale;
- $\boldsymbol{\Sigma} \in \mathbb{R}^{m \times n}$ est diagonale par blocs, avec des coefficients nuls sauf ceux de la diagonale $\{[\boldsymbol{\Sigma}]_{ii}\}_i$ qui sont positifs (ou nuls). Ces éléments, notés $\{\sigma_i\}$, s'appellent les **valeurs singulières de $\boldsymbol{A}$**.

Théorie de la SVD (3/3)

Théorème

Toute matrice $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ admet une **décomposition en valeurs singulières** (*SVD*) de la forme

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{T}},$$

où

- $\boldsymbol{U} \in \mathbb{R}^{m \times m}$ est orthogonale;
- $\boldsymbol{V} \in \mathbb{R}^{n \times n}$ est orthogonale;
- $\boldsymbol{\Sigma} \in \mathbb{R}^{m \times n}$ est diagonale par blocs, avec des coefficients nuls sauf ceux de la diagonale $\{[\boldsymbol{\Sigma}]_{ii}\}_i$ qui sont positifs (ou nuls). Ces éléments, notés $\{\sigma_i\}$, s'appellent les **valeurs singulières de $\boldsymbol{A}$**.

- $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{T}} \Leftrightarrow \boldsymbol{U}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{V}^{\mathrm{T}} = \boldsymbol{\Sigma}$;
- Une *SVD* n'est pas définie de façon unique mais ses valeurs singulières le sont.

Retour au projet : partie 3 (SVD)

Implémentation de la SVD

- Technique (relativement) basique;
- Repose sur des factorisations QR pour construire les facteurs $\boldsymbol{U}$ et $\boldsymbol{V}$.

Outils algorithmiques

- Rotations de Givens;
- Transformations de Householder;
- Matrices de pivotage.

Construction progressive de l'algorithme

Rotations de Givens

$$\begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix}$$

- On les représente via les coefficients c et s, calculés de sorte que $c^2 + s^2 = 1$ et $\begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} = \begin{bmatrix} * \\ 0 \end{bmatrix}$.
  Ex) $x = 1, z = 2 \ \Rightarrow (c = -0.4472, s = 0.8944)$.
- Les rotations que nous utilisons seront sur des colonnes consécutives.

Construction progressive de l'algorithme (2)

Transformations de Householder

$$\boldsymbol{P} = \boldsymbol{I} - \beta \boldsymbol{v}\boldsymbol{v}^\top \quad \text{telle que} \quad \boldsymbol{v}_1 = 1, \boldsymbol{P}\boldsymbol{x} = \boldsymbol{e}^1 = [10 \cdots 0]^\top .$$

- Calcul de $(\beta, \boldsymbol{v})$ à partir de $\boldsymbol{x}$, pas de construction explicite de $\boldsymbol{P}$;
- Utilisation de la structure `Vecteur`.

- $\boldsymbol{x} = \begin{bmatrix} -1 \\ 0 \end{bmatrix} \Rightarrow \left(\boldsymbol{v} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \beta = 2 \right);$
- $\boldsymbol{x} = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix} \Rightarrow \left(\boldsymbol{v} = \begin{bmatrix} 1 \\ -2.41421 \end{bmatrix}, \beta = 0.292893 \right);$
- $\boldsymbol{x} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \Rightarrow \left(\boldsymbol{v} = \begin{bmatrix} 1 \\ -1.61803 \end{bmatrix}, \beta = 0.552786 \right);$
- $\boldsymbol{x} = [-4] \Rightarrow (\boldsymbol{v} = [1], \beta = 2).$

Construction progressive de l'algorithme (3)

Factorisation QR pour matrice symétrique

Objectif : Diagonaliser $\boldsymbol{A} \in \mathbb{R}^{n \times n}$, $\boldsymbol{A}^\top = \boldsymbol{A}$.

1. Première factorisation : $\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{T}$ avec $\boldsymbol{T}$ tridiagonale ($\boldsymbol{T}_{ij} = 0$ si $j \notin \{i-1, i, i+1\}$);
2. Deuxième factorisation : diagonalisation de $\boldsymbol{T}$ de sorte à obtenir de nouveaux $\boldsymbol{Q}$ et $\boldsymbol{T}$ tels que $\boldsymbol{T}$ soit diagonale et $\boldsymbol{Q}^\top \boldsymbol{A}\boldsymbol{Q} = \boldsymbol{T}$.

Construction progressive de l'algorithme (3)

Factorisation QR pour matrice symétrique

Objectif : Diagonaliser $\boldsymbol{A} \in \mathbb{R}^{n \times n}$, $\boldsymbol{A}^\top = \boldsymbol{A}$.

1. Première factorisation : $\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{T}$ avec $\boldsymbol{T}$ tridiagonale ($\boldsymbol{T}_{ij} = 0$ si $j \notin \{i-1, i, i+1\}$);
2. Deuxième factorisation : diagonalisation de $\boldsymbol{T}$ de sorte à obtenir de nouveaux $\boldsymbol{Q}$ et $\boldsymbol{T}$ tels que $\boldsymbol{T}$ soit diagonale et $\boldsymbol{Q}^\top \boldsymbol{A}\boldsymbol{Q} = \boldsymbol{T}$.

Code d'une factorisation QR

- Les facteurs $\boldsymbol{Q}$ et $\boldsymbol{R}/\boldsymbol{T}$ sont typiquement écrits dans la matrice $\boldsymbol{A}$;
- On peut les récupérer.

Construction progressive de l'algorithme (4)

Factorisation QR non symétrique

Soit $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ avec $m \geq n$. On écrit $\boldsymbol{A\Pi} = \boldsymbol{QR}$, où :

- $\boldsymbol{\Pi}$ est une matrice de permutation;
- $\boldsymbol{Q}$ est orthogonale;
- $\boldsymbol{R}$ est triangulaire supérieure.

- Manipulation d'objets `Matrice` et `Vecteur`;
- L'objectif est d'obtenir $\boldsymbol{Q}$ et $\boldsymbol{\Pi}$.

Algorithme de la SVD complet

Entrée : $\mathbf{A} \in \mathbb{R}^{m \times n}$ *avec* $m \geq n$.

1. Calculer $\mathbf{Q}_1 \in \mathbb{R}^{n \times n}$ telle que $\mathbf{Q}_1^\top \mathbf{A}^\top \mathbf{A} \mathbf{Q}_1$ soit diagonale (factorisation QR symétrique);
2. Calculer $\mathbf{Q}_2 \in \mathbb{R}^{m \times m}$, $\mathbf{\Pi} \in \mathbb{R}^{n \times n}$ et $\mathbf{R} \in \mathbb{R}^{m \times n}$ telles que $\mathbf{Q}_2^{\mathrm{T}}(\mathbf{A}\mathbf{Q}_1)\mathbf{\Pi} = \mathbf{R}$ (factorisation avec pivot);

Sorties : $\mathbf{U} = \mathbf{Q}_2, \mathbf{\Sigma} = \mathbf{R}$ et $\mathbf{V} = \mathbf{Q}_1 \mathbf{\Pi}$.

En pratique

- Entrées/Sorties (objets `Matrice` modifiables);
- Cas $n > m$: procédure appliquée à $\mathbf{A}^\top$!

Rappel : tenseurs d'ordre d

$$\mathcal{T} = [\mathcal{T}_{i_1,i_2,\dots,i_d}]_{1\le i_1\le n_1,\dots,1\le i_d\le n_d}\,.$$

Écriture vectorielle

- Long vecteur de $n_1\, n_2 \cdots n_d$ éléments;
- L'élément $\mathcal{T}_{i_1,i_2,\dots,i_d}$ se trouve en position

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

dans le vecteur.

Partie 4 : Tenseurs d'ordre arbitraire (2)

Modes d'un tenseur $\mathcal{T} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$

Pour tout $1 \leq k \leq d$, on définit $\mathcal{T}^{(k)} \in \mathbb{R}^{n_k \times (N/n_k)}$ (où $N = \prod_{i=1}^d n_i$) via

$$\mathcal{T}_{i_1,\cdots,i_{k-1},i_k,i_{k+1},\cdots,i_d} \; = \; \mathcal{T}^{(k)}_{i_k,j_k},$$

avec $j_k = \varphi_{n_1,\cdots,n_{k-1},n_{k+1},\cdots,n_d}(i_1,\ldots,i_{k-1},i_{k+1},\ldots,i_d)$.

Partie 4 : Tenseurs d'ordre arbitraire (2)

Modes d'un tenseur $\mathcal{T} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$

Pour tout $1 \le k \le d$, on définit $\mathcal{T}^{(k)} \in \mathbb{R}^{n_k \times (N/n_k)}$ (où $N = \prod_{i=1}^d n_i$) via

$$\mathcal{T}_{i_1,\cdots,i_{k-1},i_k,i_{k+1},\cdots,i_d} = \mathcal{T}^{(k)}_{i_k,j_k},$$

avec $j_k = \varphi_{n_1,\cdots,n_{k-1},n_{k+1},\cdots,n_d}(i_1,\dots,i_{k-1},i_{k+1},\dots,i_d)$.

Produit modal

Soit $\mathcal{T} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$, $1 \le k \le d$, et $\boldsymbol{M} \in \mathbb{R}^{m_k \times n_k}$.
Le produit k-modal de $\boldsymbol{M}$ et $\mathcal{T}$, noté $\mathcal{T} \times_k \boldsymbol{M}$, 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 :

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

Partie 4 : Tenseurs d'ordre arbitraire (3)

Les tenseurs en C++

- Une classe `Tenseur` qui utilise les classes `Vecteur` et `Matrice`;
- Membres données :
  - Ordre du tenseur, tableau des dimensions;
  - Forme vectorielle du tenseur;
- Constructeurs, forme canonique;
- Surcharge d'opérateurs `+`,`-`, `[]`;
- Calcul de mode comme objet de type `Matrice`;
- Calcul de produit modal : renvoie un `Tenseur` obtenu à partir d'un `Tenseur` et d'une `Matrice`.

Partie 5 : HOSVD

Décomposition pour un tenseur $\mathcal{T} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$

- **Idée :** généraliser la SVD pour pouvoir synthétiser/approcher l'information du tenseur.
- **Outils :** les SVD de chaque mode du tenseur

$$\forall 1 \leq k \leq d,\ \mathcal{T}^{(k)} \ = \ \boldsymbol{U}^{(k)} \boldsymbol{\Sigma}^{(k)} \left[\boldsymbol{V}^{(k)}\right]^{\mathrm{T}}.$$

SVD d'ordre élevé (HOSVD)

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

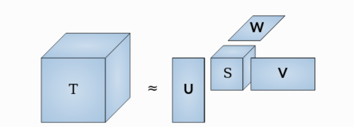
où

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

est appelé le *coeur de tenseur*.

Exemple de HOSVD (Source: tensorlab.com)

- Décomposition d'un tenseur d'ordre 3 :



- Dernière question du projet : décomposer un tenseur d'ordre 3 de taille $3 \times 3 \times 3$ de mode 1 :

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

- Il faudra retrouver le tenseur originel à 10^{-3} près.

Sommaire

1. Projet - Point de vue théorique
2. Projet - Point de vue implémentation
3. Questions/Réponses

Partie 1 du projet : les vecteurs

En maths : tableau à une entrée

$$\boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{x} = [\boldsymbol{x}_i]_{i=1,\dots,n} = \begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_n \end{bmatrix}.$$

Pour le projet C++

- Classe `Vecteur` contenant un tableau de réels flottants et sa dimension ⇒ Allocation dynamique
- Plusieurs constructeurs + Forme canonique;
- Opérateurs à surdéfinir : `+`, `-`, `[]`
- Norme et produit scalaire :

$$\boldsymbol{x}^\top \boldsymbol{y} = \sum_{i=1}^{n} \boldsymbol{x}_i \boldsymbol{y}_i, \quad \|\boldsymbol{x}\| = \sqrt{\boldsymbol{x}^\top \boldsymbol{x}}.$$

Conseils : partie 1

- Validation (très) incrémentale : tout le reste du projet dépend de cette partie;
- La forme canonique doit être bien implémentée;
- Un constructeur sans argument sera utile pour la suite;
- Opérateur [ ] : permet d'accéder et **de modifier** les coefficients des vecteurs.

Validation de la partie 1

- Vérifier que chaque constructeur fonctionne;
- Accéder aux composantes et les modifier.

Et après...

- Les classes `Matrice` et `Tenseur` utilisent des objets de type `Vecteur`;
- Il faudra donc les déclarations d'amitié appropriées.

Partie 2 du projet : les matrices

En maths : Tableau à deux entrées

$$\boldsymbol{A} \in \mathbb{R}^{m \times n}, \quad \boldsymbol{A} = [\boldsymbol{A}_{ij}]_{\substack{1 \le i \le m \\ 1 \le j \le n}} = \begin{bmatrix} \boldsymbol{A}_{1,1} & \cdots & \boldsymbol{A}_{1,n} \\ \vdots & \ddots & \vdots \\ \boldsymbol{A}_{m,1} & \cdots & \boldsymbol{A}_{m,n} \end{bmatrix}.$$

Pour le projet C++

- Un tableau d'objets de type `Vecteur` (colonnes de la matrice);
- Les dimensions de la matrice;
- Constructeurs, forme canonique;
- Surdéfinitions : `+`,`-`,`[ ]`,`*`
  (*Produit matriciel :* Si $\boldsymbol{C} = \boldsymbol{A} * \boldsymbol{B}$ avec $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ et $\boldsymbol{B} \in \mathbb{R}^{n \times p}$, alors $\forall i = 1..m, \forall j = 1..p,\ \boldsymbol{C}_{i,j} = \sum_{k=1}^{n} \boldsymbol{A}_{i,k} \boldsymbol{B}_{k,j}$.)
- *Norme de Frobenius :* $\|\boldsymbol{A}\|_F^2 = \sum_{i,j} \boldsymbol{A}_{ij}^2$.
- *Transposée :* $\left[\boldsymbol{A}^\top\right]_{ji} = \boldsymbol{A}_{ij}$.

Conseils pour la partie 2

- Attention à l'accès aux coefficients :

```
Matrice M (2,3); // 2 lignes/3 colonnes
M[2][0] = 1; //1re ligne, 3e colonne
```

- Problèmes d'allocation dynamique plus sophistiqués que `Vecteur`;
- La classe `Matrice` sera utilisée par la classe `Tenseur`.

- Certaines instructions auront besoin d'une boucle simple ou double, ou d'une fonction dédiée (deux solutions valides);
- Ex) Mise à jour d'une sous-matrice.

L'algorithme

- Version simplifiée de la méthode de référence;
- Décomposé en plusieurs routines standards, cf [Golub and Van Loan, 2013];
- **Outil-clé :** décompositions matricielles.

Conseils

- Plusieurs algorithmes $\Rightarrow$ validation incrémentale;
- Un seul type de retour possible $\Rightarrow$ passage par **référence**;
- Lors de la validation, affichez les résultats au fur et à mesure pour valider.

Rotations de Givens

$$\begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix}$$

- On les représente via les coefficients c et s, calculés de sorte que $c^2 + s^2 = 1$ et $\begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} = \begin{bmatrix} * \\ 0 \end{bmatrix}$.
  Ex) $x = 1, z = 2 \Rightarrow (c = -0.4472, s = 0.8944)$.
- Les rotations que nous utilisons seront sur des colonnes consécutives.

Transformations de Householder

$$\boldsymbol{P} = \boldsymbol{I} - \beta \boldsymbol{v}\boldsymbol{v}^\top \quad \text{telle que} \quad \boldsymbol{v}_1 = 1, \boldsymbol{P}\boldsymbol{x} = \boldsymbol{e}^1 = [1 \; 0 \; \cdots \; 0]^\top.$$

- Calcul de $(\beta, \boldsymbol{v})$ à partir de $\boldsymbol{x}$, pas de construction explicite de $\boldsymbol{P}$;
- Utilisation de la structure `Vecteur`.

- $\boldsymbol{x} = \begin{bmatrix} -1 \\ 0 \end{bmatrix} \Rightarrow \left(\boldsymbol{v} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \; \beta = 2\right)$;
- $\boldsymbol{x} = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix} \Rightarrow \left(\boldsymbol{v} = \begin{bmatrix} 1 \\ -2.41421 \end{bmatrix}, \; \beta = 0.292893\right)$;
- $\boldsymbol{x} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \Rightarrow \left(\boldsymbol{v} = \begin{bmatrix} 1 \\ -1.61803 \end{bmatrix}, \; \beta = 0.552786\right)$;
- $\boldsymbol{x} = [-4] \Rightarrow (\boldsymbol{v} = [1], \; \beta = 2)$.

Conseils Partie 3/Algorithmes 3-4

Algorithme 3 : Réduction tridiagonale

- Attention au passage par référence (modification des entrées);
- Attention aux indices !

Algorithme 4 : Factorisation QR symétrique

- Astuces numériques : Mettre à 0 certains coefficients, symétriser;
- Attention aux indices !
- Vérifier l'orthogonalité de $\boldsymbol{Q}$: $\boldsymbol{Q}\boldsymbol{Q}^\top$ doit être (proche de) la matrice identité.

Factorisation QR pour matrice symétrique

Objectif : Diagonaliser $\boldsymbol{A} \in \mathbb{R}^{n \times n}$, $\boldsymbol{A}^\top = \boldsymbol{A}$.

1. Première factorisation : $\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{T}\boldsymbol{Q}^\top$ avec $\boldsymbol{T}$ tridiagonale ($\boldsymbol{T}_{ij} = 0$ si $j \notin \{i-1, i, i+1\}$);
2. Deuxième factorisation : diagonalisation de $\boldsymbol{T}$ de sorte à obtenir de nouveaux $\boldsymbol{Q}$ et $\boldsymbol{T}$ tels que $\boldsymbol{T}$ soit diagonale et $\boldsymbol{Q}^\top \boldsymbol{A}\boldsymbol{Q} = \boldsymbol{T}$.

Validation 1 des algorithmes

Pour la matrice $\boldsymbol{A} = \begin{bmatrix} 10 & -6 \\ -6 & 10 \end{bmatrix}$, il faut obtenir :

$$\boldsymbol{Q} = \begin{bmatrix} 0.707107 & 0.707107 \\ -0.707107 & 0.707107 \end{bmatrix}, \quad \boldsymbol{T} = \begin{bmatrix} 16 & 0 \\ 0 & 4 \end{bmatrix};$$

Validation 2 des algorithmes

Pour la matrice $\boldsymbol{A} = \begin{bmatrix} 10 & -6 & 0 \\ -6 & 10 & 0 \\ 0 & 0 & 1 \end{bmatrix}$.

En phase 1 de l'algorithme (avant appel à l'algorithme 3) :

$$\boldsymbol{Q} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{T} = \begin{bmatrix} 10 & -6 & 0 \\ -6 & 10 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

En phase 2 de l'algorithme :

$$\boldsymbol{Q} = \begin{bmatrix} 0.707107 & -0.707107 & 0 \\ -0.707107 & -0.707107 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{T} = \begin{bmatrix} 16 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Factorisation QR non symétrique

Soit $\mathbf{A} \in \mathbb{R}^{m \times n}$ avec $m \geq n$. On écrit $\mathbf{A\Pi} = \mathbf{QR}$, où :

- $\mathbf{\Pi}$ est une matrice de permutation;
- $\mathbf{Q}$ est orthogonale;
- $\mathbf{R}$ est triangulaire supérieure.

- $\mathbf{Q}$ et $\mathbf{\Pi}$ seront passées par référence;
- Attention aux divers indices de boucle.

Matrice de départ :

$$\boldsymbol{A} = \begin{bmatrix} 1 & 1/2 & 1/3 \\ 1/2 & 1/3 & 1/4 \\ 1/3 & 1/4 & 1.5 \end{bmatrix}$$

Décomposition :

$$\boldsymbol{Q} = \begin{bmatrix} 0.857143 & -0.496929 & -0.135526 \\ 0.428571 & 0.542105 & 0.722806 \\ 0.285714 & 0.677631 & -0.677631 \end{bmatrix},$$

$$\boldsymbol{R} = \begin{bmatrix} 1.16667 & 0.45 & 0.642857 \\ 0 & 0.105409 & 0.101645 \\ 0 & 0 & 0.00376463 \end{bmatrix},$$

$$\boldsymbol{\Pi} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Validation finale de la partie 3

- $\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix};$
- $\mathbf{B} = \begin{bmatrix} 2\sqrt{2} & -2\sqrt{2} \\ -\sqrt{2} & -\sqrt{2} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 4 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} \sqrt{2}/2 & -\sqrt{2}/2 \\ \sqrt{2}/2 & \sqrt{2}/2 \end{bmatrix};$
- $\mathbf{C} = \begin{bmatrix} 1/2 & 3\sqrt{3}/2 & 0 \\ \sqrt{3}/2 & -3/2 & 0 \end{bmatrix} = \begin{bmatrix} \sqrt{3}/2 & 1/2 \\ -1/2 & \sqrt{3}/2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 3 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$

Partie 4 : Tenseurs d'ordre arbitraire

- Une classe `Tenseur` qui utilise les classes `Vecteur` et `Matrice`;
- Membres données :
  - Ordre du tenseur, tableau des dimensions;
  - Forme vectorielle du tenseur;
- Constructeurs, forme canonique;
- Surcharge d'opérateurs `+`,`-`,`[]`;
- Calcul de mode comme objet de type `Matrice`;
- Calcul de produit modal : renvoie un `Tenseur` obtenu à partir d'un `Tenseur` et d'une `Matrice`.

- Pas besoin de la partie 3, mais besoin des classes des parties 1 et 2;
- Attention aux indices dans la fonction φ;
- Recommandé : créer une fonction φ, ainsi que son inverse et son équivalent pour un mode;

```
int phi(int ordre, int *dims,int *indices){};
void invphi(int ordre, int *dims, int i, int *indices){};
int phik(int ordre, int *dims,int *indices,int mode){};
```

Validation de la partie 4

Tenseur $\mathcal{T} \in \mathbb{R}^{2\times 2\times 2}$ tel que

$$\mathcal{T}^{(2)} = \begin{bmatrix} 1 & 3 & 0 & -1 \\ 4 & 1/3 & 1.5 & 2 \end{bmatrix},$$

matrice $\boldsymbol{A} = \begin{bmatrix} 3 & -1 \\ 0 & 6 \\ 0 & -3 \end{bmatrix}$.

Calcul de $\mathcal{S} = \mathcal{T} \times_3 \boldsymbol{A}$: on a

$$\mathcal{S}^{(3)} = \begin{bmatrix} 0 & 35/3 & 1 & 5/2 \\ 18 & 2 & -6 & 12 \\ -9 & -1 & 3 & -6 \end{bmatrix}.$$

Décomposition pour un tenseur $\mathcal{T} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$

- **Idée :** généraliser la SVD pour pouvoir synthétiser/approcher l'information du tenseur.
- **Outils :** les SVD de chaque mode du tenseur

$$\forall 1 \leq k \leq d, \ \mathcal{T}^{(k)} \ = \ \boldsymbol{U}^{(k)} \boldsymbol{\Sigma}^{(k)} \left[\boldsymbol{V}^{(k)} \right]^{\mathrm{T}}.$$

SVD d'ordre élevé (HOSVD)

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

où

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

est appelé le *coeur de tenseur*.

Conseils : partie 5

Classe `TenseurSVD`

- Les membres de la classe `Tenseur` doivent être accessibles dans la classe dérivée `TenseurSVD` $\Rightarrow$ `protected`;
- Appel implicite au destructeur de `Tenseur` dans celui de `TenseurSVD`;
- Un constructeur de `TenseurSVD` doit faire appel à un constructeur de `Tenseur`;
- Point-clé : manipulation du tableau d'objets de type `Matrice`.

La fonction `hosvd`

- Besoin d'un constructeur pour `TenseurSVD`;
- Le code doit être court grâce aux fonctions `svd` et `pmod`.

Dernière question du projet : décomposer un tenseur $\mathcal{T}$ d'ordre 3 de taille $3 \times 3 \times 3$ de mode 1

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

- **But :** Obtenir $\mathcal{S}, \boldsymbol{U}_1, \boldsymbol{U}_2, \boldsymbol{U}_3$ tels que

$$\mathcal{T} \approx \tilde{\mathcal{T}} = \mathcal{S} \times_1 \boldsymbol{U}_1 \times_2 \boldsymbol{U}_2 \times_3 \boldsymbol{U}_3$$

- Vérification de l'erreur relative :

$$\max_{\substack{1 \le i \le 3 \\ 1 \le j \le 9}} \frac{\left| \left[\mathcal{T}^{(1)} - \tilde{\mathcal{T}}^{(1)} \right]_{ij} \right|}{\| \mathcal{T}^{(1)} \|_F} \le 10^{-3}.$$

Sommaire

1. Projet - Point de vue théorique
2. Projet - Point de vue implémentation
3. Questions/Réponses

Annexe : Questions/Réponses

- Quelques questions reçues durant le projet;
- S'enrichiront au fur et à mesure.

Q& A : initialisation de tableaux et pointeurs (1/2)

Classes Vecteur et Matrice

```
class Vecteur{
  // ...
  Vecteur();
  Vecteur(int);
};

class Matrice{
  Vecteur *cols;
  // ...
  Matrice();
  Matrice(int,int);
};
```

- Comment initialiser `cols` dans les constructeurs de la classe `Matrice` ?
- Comment allouer la mémoire nécessaire ?

Q& A : initialisation de tableaux et pointeurs (2/2)

```
Matrice::Matrice(){
  cols = nullptr;
}

Matrice::Matrice(int nblignes,int nbcolonnes){
  // ...
  cols = new Vecteur[nbcolonnes];
}
```

- Pointeur nul pour une initialisation (pas de mémoire à allouer ici);
- Opérateur `new [ ]` pour un tableau
  Attention : requiert un constructeur sans argument.

Q& A : Surcharge d'opérateur binaire

```
class Matrice{
  // ...
  Matrice operator +(Matrice);

  friend operator *(float,Matrice);

};
```

- L'opérateur + est **binaire** : son premier argument sera l'objet appelant de par sa déclaration;
- L'opérateur $*$ est aussi binaire, mais déclaré en dehors de la classe : ses deux arguments sont spécifiés.

- Classe `Vecteur` du projet : implémente des vecteurs colonnes;
- Classe `Matrice` : implémente des matrices, y compris à une ligne et/ou une colonne.

Problème : calcul de $\mathbf{p}^\top \mathbf{v}\mathbf{v}$, où $\mathbf{p}, \mathbf{v} \in \mathbb{R}^n$

- Point de vue maths : $\mathbf{p}^\top \in \mathbb{R}^{1 \times n}$;
- Point de vue numérique : le meilleur choix est de considérer $\mathbf{p}^\top \mathbf{v} \in \mathbb{R}$ $\Rightarrow$ fonction `dot`.

Q & A : Création de tableau dynamique

Contexte

- Le projet utilise de nombreux tableaux;
- La taille de ces tableaux est a priori inconnue
  $\Rightarrow$ Utilisation de pointeurs.

Allocation dynamique

```
int m;
int *tab;
cin>>m;
tab = new int[m];
```

- Valeur de `m` inconnue à la compilation $\Rightarrow$ `tab` doit être déclaré comme un pointeur !

Q & A : Manipulation de tableau dynamique

- En paramètre d'appel : pas de recopie;
- En paramètre de retour : déconseillé.

```
myfun(int * t,int b,int *u){ u=t;}

myfun2(int *t,int b,int *u){
  int *aux= new int[b];
  // Copie de t dans aux
  u=aux;
}

int d=3;
int t[d]={3,3,3};

myfun(t,d,u);
u[0];//Acces autorise
myfun2(t,d,u);
u[0];//Erreur
```

Q & A : Destructeur et allocation dynamique

Lors d'un appel de destructeur

- Destruction d'une partie allouée dynamiquement : `delete` ou `delete [ ];`
- Appel des destructeurs des objets membres.

Exemples

```
Vecteur::~Vecteur(){ delete [] tab;}

Matrice::~Matrice(){
  delete [] mat; // Appel au destructeur de Vecteur
}
```

Q & A : Déclaration d'amitié de classes

Contexte

- Classe `Tenseur` doit être amie de la classe `Vecteur`;
- Classe `Vecteur` déclarée avant/dans un autre fichier.

Dans le fichier de la classe `Vecteur` (ex: Vecteur.h) :

```
class Tenseur;// Declaration

class Vecteur{
  // ...
};
```

Dans le fichier de la classe `Tenseur` :

```
#include "Vecteur.h"

class Tenseur{
  // ...
};
```

En résumé

À garder en tête

- Projet long $\Rightarrow$ validez petit à petit;
- Parties 1, 2, 4 les plus fondamentales;
- Attention à l'allocation dynamique;
- Les indices commencent à 0 en C++.

Rappel des modalités

- Date limite d'envoi : 5 juin 2020;
- Par mail : `clement.royer@dauphine.psl.eu`;
- 1 envoi par étudiant.