Analyse Multivariée – Sujet du BE
S8 IATI EEEA – ENSEEIHT

Enseignant responsable : Henrique Goulart

Année 2024–2025

[version : 12 mars 2025]

Exercice 1 : Axes et composantes principaux

Soient X_1 et X_2 deux variables indépendantes et telles que $\mathbb{E}\{X_1\} = \mathbb{E}\{X_2\} = 0$ et $\mathrm{Var}(X_1) = \sigma_1^2$, $\mathrm{Var}(X_2) = \sigma_2^2$.

1) Quelles sont les axes principaux au sens de la PCA pour le vecteur $\underline{X} = (X_1, X_2)^\mathsf{T}$ et les composantes principales associées ?

2) Tracer $n = 400$ réalisations de $\underline{X}$ (en nuage de points), avec X_1 distribué selon la loi uniforme et X_2 distribué selon la loi normale, avec
   — $\sigma_1^2 = \sigma_2^2 = 1$;
   — $\sigma_1^2 = 4$, $\sigma_2^2 = 1$.
   On utilisera à chaque fois les moments donnés pour déterminer les paramètres des lois de X_1 et X_2. Est-on capable de trouver les axes principaux au sens de la PCA par inspection visuelle des données dans chacun des cas ?

   Indication : Pour rappel, si $X \sim \mathcal{U}([a, b])$, alors $\mathrm{Var}(X) = \frac{(b-a)^2}{12}$.

3) Répéter l'item 1 pour le vecteur aléatoire donné par

$$\underline{Y} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \underline{X}.$$

   Ensuite, générer des réalisations de $\underline{Y}$ avec $\theta = \pi/4$ à partir de celles de $\underline{X}$ générées dans l'item 2), et les tracer en nuage de points. Commenter les résultats.

Exercice 2 : SVD et approximation de rang faible

Dans cet exercice, on utilisera la SVD afin de compresser des images. Charger les images `image_1_gray.jpg` et `image_2_gray.jpg` disponibles sur Moodle, à l'aide de la fonction `imread` de `matplotlib.pyplot`. La procédure suivante devra être réalisée pour chacune de ces images, que nous allons traiter comme des matrices $\mathbf{X} \in \mathbb{R}^{p \times n}$, dont les composantes représentent les niveaux de gris des pixels.

1) Après centrer chacune des lignes de l'image par

$$\tilde{\mathbf{X}} = \mathbf{X} - \frac{1}{n} \mathbf{X} \underline{1}_n \underline{1}_n^\mathsf{T}, \tag{1}$$

on calculera la meilleure approximation de rang r de $\tilde{\mathbf{X}}$, notée $\tilde{\mathbf{X}}_r$, à l'aide de sa SVD pour $r = 20, 50, 100$.

2) En utilisant la fonction `imshow` de `matplotlib.pyplot`, afficher chacune des approximations $\tilde{\mathbf{X}}_r$ avec la moyenne des lignes restituée, c'est-à-dire, $\tilde{\mathbf{X}}_r + \frac{1}{n}\mathbf{X}\,\underline{1}_n\,\underline{1}_n^{\mathsf{T}}$.

3) Tracer l'erreur quadratique normalisée de l'approximation de chaque image par sa SVD tronquée à r composantes, avec $r = 0, 1, \ldots, 200$:

$$f(r) := \frac{\left\|\tilde{\mathbf{X}} - \tilde{\mathbf{X}}_r\right\|_{\mathsf{F}}^2}{\left\|\tilde{\mathbf{X}}\right\|_{\mathsf{F}}^2}. \tag{2}$$

4) Commenter les résultats obtenus dans les items précédents.

Exercice 3 : PCA pour réduction de dimension

On pose

$$\underline{X} = \mathbf{A}\,\underline{Z} + \sigma\,\underline{B}, \tag{3}$$

avec $\underline{Z}$ indépendant de $\underline{B}$,

$$\mathbf{A} = \begin{pmatrix} 1 & \alpha \\ 0 & \sqrt{1-\alpha^2} \end{pmatrix}, \qquad \alpha \in]-1, 1[, \tag{4}$$

et $\underline{Z} \sim \mathcal{N}(\underline{0}, \mathbf{I}_2)$, $\underline{B} \sim \mathcal{N}(\underline{0}, \mathbf{I}_2)$.

Le terme $\mathbf{A}\,\underline{Z}$ du modèle (3) représente le signal d'intérêt, alors que $\sigma\,\underline{B}$ représente le bruit. L'objectif est alors d'exploiter cet exemple en petite dimension afin de comprendre ce qui se passe lorsque nous employons la PCA pour réduire la dimension des données (ici donc une projection sur un axe unidimensionnel), en fonction de la distribution de celles-ci dans l'espace. Concrètement, en faisant varier α, nous pouvons changer la forme du nuage de points, ce qui se répercutera sur la qualité de l'approximation de ces points par leur projection sur le premier axe principal.

1) D'après l'expression de la matrice $\mathbf{A}$, que se passe-t-il avec l'angle entre ses colonnes lorsque $\alpha \to 0$? Et lorsque $\alpha \to \pm 1$?

2) Écrire un programme qui génère $n = 200$ réalisations de $\underline{X}$, d'abord avec $\sigma^2 = 0$ (c'est-à-dire, sans bruit), et pour

$$\alpha = 0.01,\ 0.25,\ 0.5,\ 0.75,\ 0.99.$$

Tracer à chaque fois le nuage de points généré (à l'aide de la commande `matplotlib.pyplot.scatter`) et les vecteurs propres de $\mathbf{C}$ multipliés par les respectives valeurs propres, $\lambda_1\underline{u}_1$ et $\lambda_2\underline{u}_2$. Ensuite, répéter pour $\sigma^2 = 0.4$ et puis $\sigma^2 = 0.8$. Qu'observez-vous ?

Indication : On pourra utiliser la fonction `matplotlib.pyplot.quiver` pour tracer les vecteurs.

3) Enfin, on évaluera ici la qualité de l'approximation obtenue par projection sur le premier axe principal. Calculer la première composante principale des données pour chaque valeur de α, en effectuant la projection sur $\underline{u}_1$, d'abord

avec $\sigma^2 = 0$. Calculer et tracer en fonction de α l'erreur relative moyenne d'approximation définie comme :

$$\hat{\epsilon}(\alpha) := 1 - \frac{\sum_{i=1}^{n} (\underline{u}_1^{\mathsf{T}} \underline{x}_i)^2}{\sum_{i=1}^{n} \|\underline{x}_i\|^2}.$$

Ensuite, on va la comparer à la quantité théorique

$$\epsilon(\alpha) := 1 - \frac{\mathbb{E}\left\{(\underline{u}_1^{\mathsf{T}} \underline{X})^2\right\}}{\mathbb{E}\left\{\|\underline{X}\|^2\right\}} = \frac{1 + \sigma^2 - \alpha}{2\,(1 + \sigma^2)}.$$

Répéter pour $\sigma^2 = 0.4$ et puis $\sigma^2 = 0.8$. (Tracer toutes les courbes dans une même figure.) Quelles sont vos conclusions ?

Exercice 4 : FastICA par kurtosis

L'algorithme ICA pour l'extraction d'une composante indépendante basée sur la kurtosis cherche une solution à

$$\max_{\|\underline{q}\|=1} |\mathrm{kurt}(Y)| = \max_{\|\underline{q}\|=1} \left|\mathrm{kurt}(\underline{q}^{\mathsf{T}} \underline{Z})\right|,$$

où $\underline{Z} = \mathbf{W}\underline{X} = \tilde{\mathbf{A}}\underline{S}$, avec $\tilde{\mathbf{A}} \in \mathbb{O}(p)$. Comme $\mathrm{Cov}\,(\underline{Z}) = \mathbf{I}$ (car $\underline{Z}$ est le vecteur obtenu après blanchiment), on sait de plus que $\mathrm{Var}\,(Y) = \|\underline{q}\|^2 = 1$.

Dans cet exercice, on écrira un programme pour réaliser l'extraction d'une composante indépendante S du mélange, une fois effectué le blanchiment des données. On cherchera donc à estimer un vecteur $\underline{q} \in \mathbb{R}^p$ de norme unité sur lequel on pourra projeter ces données afin de calculer les réalisations de S.

Le programme prendra comme entrée une matrice $\mathbf{Z}$ contenant les observations blanchies, une solution initiale $\underline{q}^{(0)}$, et une tolérance ϵ pour le critère d'arrêt. A chaque itération, on mettra à jour $\underline{q}^{(k)}$ par l'approche de maximisation de la valeur absolue de la kurtosis. Ce vecteur devra être toujours normalisé comme dans l'algorithme vu en cours. Comme critère d'arrêt, on utilisera l'inégalité

$$1 - \left|\left\langle \underline{q}^{(k+1)}, \underline{q}^{(k)} \right\rangle\right| < \epsilon,$$

qui se vérifie lorsque deux vecteurs estimés consécutifs sont suffisamment alignés.

Tester votre programme pour $p = 2$, avec $S_1, S_2 \overset{\text{i.i.d.}}{\sim} \mathcal{U}([-\sqrt{3}, \sqrt{3}])$,

$$\mathbf{A} = \begin{pmatrix} 1 & 2 \\ 4 & 3 \end{pmatrix}$$

et $\epsilon = 10^{-6}$. Le vecteur $\underline{q}^{(0)}$ pourra être généré aléatoirement (n'oubliez pas de le normaliser). Après convergence, afficher les données en nuage de points, ainsi que le vecteur estimé (à l'aide de la fonction `quiver` de `pyplot`). Répéter plusieurs fois cette expérience. Que concluez-vous en ce qui concerne la direction estimée et le nombre d'itérations nécessaires ?

Exercice bonus : Projection sur $\mathbb{O}(p)$

La variante parallèle de l'algorithme FastICA pour la maximisation de la non-gaussianité consiste à estimer les colonnes de la matrice de projection $\mathbf{Q}$ en les mettant à jour en parallèle et puis en les rendant orthogonales à l'aide d'une projection basée sur la SVD, comme vu en cours. Dans cet exercice, nous allons étudier deux autres méthodes pour projeter une matrice $\tilde{\mathbf{Q}} \in \mathbb{R}^{p\times p}$ sur $\mathbb{O}(p)$:

1) La première méthode consiste à calculer

$$\mathbf{Q} = \tilde{\mathbf{Q}} \left(\tilde{\mathbf{Q}}^\mathsf{T}\tilde{\mathbf{Q}}\right)^{-1/2},$$

où la racine d'une matrice symétrique définie positive $\mathbf{M} \in \mathbb{R}^{p\times p}$ est définie par

$$\mathbf{M}^{-1/2} := \mathbf{V}\,\mathrm{Diag}(\lambda_1^{-1/2}, \ldots, \lambda_p^{-1/2})\mathbf{V}^\mathsf{T},$$

avec $\mathbf{M} = \mathbf{V}\,\mathrm{Diag}(\lambda_1, \ldots, \lambda_p)\mathbf{V}^\mathsf{T}$ la décomposition spectrale de $\mathbf{M}$.

2) Dans la seconde méthode, on réalise une projection approchée sans calculer une décomposition spectrale ou SVD, à l'aide de l'algorithme itératif suivant. On initialise $\mathbf{Q}^{(0)} := \tilde{\mathbf{Q}} \left\|\tilde{\mathbf{Q}}\right\|^{-1}$ afin d'avoir une solution initiale $\mathbf{Q}^{(0)}$ dont toutes les valeurs singulières sont dans $[0, 1]$, et puis on calcule à chaque itération k

$$\mathbf{Q}^{(k+1)} = \frac{3}{2}\mathbf{Q}^{(k)} - \frac{1}{2}\mathbf{Q}^{(k)} \left(\mathbf{Q}^{(k)}\right)^\mathsf{T} \mathbf{Q}^{(k)},$$

jusqu'à ce que $\left(\mathbf{Q}^{(k+1)}\right)^\mathsf{T} \mathbf{Q}^{(k+1)} \approx \mathbf{I}$. Par exemple, on arrête l'algorithme lorsque $\left\|\mathbf{I} - \left(\mathbf{Q}^{(k+1)}\right)^\mathsf{T} \mathbf{Q}^{(k+1)}\right\| < \epsilon$ pour une tolérance $\epsilon > 0$ choisie.

Écrire un programme pour implémenter les deux méthodes ci-dessus, et comparer leurs résultats pour une matrice $\mathbf{Q}^{(0)} \in \mathbb{R}^{p\times p}$ dont les éléments sont tirés aléatoirement de la loi $\mathcal{N}(0, 1)$. On pourra choisir $\epsilon = 10^{-4}$ comme tolérance pour l'algorithme itératif. Pour chacune des solutions $\mathbf{Q}$ obtenues, on calculera en particulier

$$E(\mathbf{Q}) = \max_{ij} \left|\left(\mathbf{Q}^\mathsf{T}\mathbf{Q}\right)_{ij} - \delta_{ij}\right|,$$

et on mesurera le temps de calcul à l'aide de la bibliothèque `time` :

```
import time

start_time = time.perf_counter()
# Code de la méthode 1
end_time = time.perf_counter()
# Calculer temps d'exécution
temps1 = end_time - start_time

start_time = time.perf_counter()
# Code de la méthode 2
end_time = time.perf_counter()
# Calculer temps d'exécution
```

```
temps2 = end_time - start_time
```

Commenter les résultats obtenus pour $p = 10$ et $p = 300$, en matière du critère de performance $E(\mathbf{Q})$ et du temps de calcul.