Situation d'apprentissage et d'évaluation Débruitage d'images par Analyse en Composantes Principales

CHOSSON Clément
GINESTE Thomas
MONDINA Antonin
THIEBAUT Mattias
VALENTE Mathias

ING1 groupe 11

suppervisé par : RANISAVLJEVIC Élisabeth,
FORTIN CAMDAVANT Nisrine,
FASSI Dieudonné

25 mai 2025

Contents

1 Introduction

Le traitement d'images est un domaine clé en informatique, avec des applications dans la photographie numérique, le médical ou la sécurité. Dans le cadre de ce projet, nous nous intéressons à une problématique classique mais toujours d'actualité : le débruitage d'images numériques.

Une image acquise par un capteur, comme une caméra ou un scanner, est souvent altérée par différents types de bruit. Le bruit que nous considérons ici est le bruit gaussien additif, modélisé comme un signal aléatoire centré, généralement issu de perturbations électroniques internes au capteur ou de conditions d'acquisition difficiles (faible lumière, température, etc.). Ce bruit est non structuré, souvent peu visible à l'œil nu, mais peut considérablement nuire à la qualité de traitement et d'analyse de l'image.

L'objectif principal de ce projet est de restaurer une image dégradée par du bruit gaussien en estimant l'image originale la plus probable. Pour cela, nous utilisons une méthode statistique puissante : l'analyse en composantes principales (ACP) ou, en anglais, Principal Component Analysis (PCA). L'ACP permet de transformer un ensemble de données corrélées (ici, les petits blocs d'image appelés patchs) en un espace où les données sont représentées de façon plus compacte et moins redondante.

Dans ce rapport, nous expliquerons la méthode mathématiques que nous allons utiliser pour débruiter l'image ainsi que l'implémentation d'une interface homme-machine (IHM). Nous analyserons ensuite les résultats obtenus selon le type de débruitage que nous appliquerons afin de déterminer une méthode optimale pour débruiter une image. Nous étudierons et comparerons deux variantes principales de l'ACP : une approche globale (PGPCA), où l'analyse est faite sur l'ensemble des patchs de l'image, et une approche locale (PLPCA), qui applique l'ACP sur des zones restreintes.

2 Réponse au problème

2.1 Approche mathématiques

2.1.1 Préambule

Pour commencer nos explications, nous allons démontrer que l'on peut décomposer un vecteur V correspondant à un patch dans une base orthonormée.

Soit $\beta = \{u_1, \ldots, u_{s^2}\}$ une base orthonormée de $\mathbb{R}^{s^2}$.
On notera ici u un vecteur de β et $u^\top$ sa transposée.

Soit $x = V(k) - m(v) \in \mathbb{R}^{s^2}$.
On a :

$$x = \sum_{i=1}^{s^2} \beta_i u_i$$

De plus :

$$\langle x \mid u_j \rangle = \left\langle \sum_{i=1}^{s^2} \beta_i u_i \middle| u_j \right\rangle = \sum_{i=1}^{s^2} \beta_i \langle u_i \mid u_j \rangle = \beta_j \langle u_j \mid u_j \rangle = \beta_j$$

Donc :

$$\beta_i = \langle x \mid u_i \rangle$$

Ainsi :

$$V(k) - m(v) = \sum_{i=1}^{s^2} \langle V(k) - m(v) \mid u_i \rangle \, u_i = \sum_{i=1}^{s^2} u_i^\top \left(V(k) - m(v)\right) u_i$$

Finalement :

$$V(k) = m(v) + \sum_{i=1}^{s^2} u_i^\top \left(V(k) - m(v)\right) u_i$$

Figure 1: Démonstration de la décomposition de V dans une base orthonormée

2.2 Transformation du problème en mathématiques

Tout d'abord, l'objectif du débruitage est de travailler sur des patchs qui sont des petits carrées de pixels extraient de l'image. Nous allons représenter ces patchs sous forme de vecteurs dont les valeurs sont des nuances de gris comprises entre $[0, 255]$. Les s pixels de la première ligne seront les s premières composantes du vecteurs, ainsi de suite.

Nous allons ensuite débruiter l'image à l'aide de l'ACP appliquée à ces vecteurs. L'ACP va transformer les observations de variables corrélées en observations de variables non corrélées, ce qui va centrer les variables principales, et ainsi rendre aberrantes les variables de bruits.

En appliquant un seuillage avec une valeur seuil, nous pourrons supprimer les variables aberrantes, et donc supprimer le bruit.

2.3 Approche informatique

Comme demandé, notre application permettant de répondre à cette problématique de bruitage est développé en Java et l'interface graphique à l'aide de JavaFX.
Pour ce qui est de la structure du code, nous avons divisé le problème en plusieurs classes :

2.3.1 ACP

Cette classe est utilisée pour appliquer l'Analyse en Composante Principale sur chaque patch extrait de l'image bruitée. Elle est composée de trois méthodes :

- public static ACPResult computeACP(List⟨double[ ]⟩ V) : applique l'ACP sur un vecteur placé en paramètre
- public static MoyCovResult MoyCov(List⟨double[ ]⟩ V) : calcule la moyenne de covariance d'un vecteur placée en paramètre
- public static double[ ][ ] project(double[ ][ ] U, List⟨double[ ]⟩ Vc) : calcule la contribution des vecteurs

2.3.2 Evaluation

Cette classe va servir à calculer les indicateurs de performances de notre débruitage au travers des indicateurs que sont le MSE et le PSNR.
Voici les méthodes de cette classe :

- public static double mse(BufferedImage original, BufferedImage denoised) : calcule le mse
- public static double psnr(double mse) : calcule le PSNR

2.3.3 ImageUtils

Cette classe va représenter l'image sur laquelle nous travaillons et va lui appliquer les différentes méthodes afin de la bruiter puis de la débruiter

- public static BufferedImage noising(BufferedImage X0, double sigma) : bruite l'image
- public static List⟨Patch⟩ extractPatches(BufferedImage X, int s) : extrait des patchs de l'image
- public static BufferedImage reconstructPatches(List⟨Patch⟩ patches, int height, int width) : reconstruit l'image à partir de sa liste de patchs
- public static List⟨ImageZone⟩ decoupeImage(BufferedImage X, int W, int n) : découpe une image en plusieurs sous images
- public static List⟨VectorWithPosition⟩ VectorPatchs(List¡Patch¿ patches) : convertit un patch en vecteur et l'enregistre avec les coordonnées du patch
- public static double computeMSE(BufferedImage img1, BufferedImage img2) : calcul le MSE entre 2 images
- public static double computePSNR(double mse) : calcul le PSNR à partir du MSE

2.3.4 Maquette

Cette classe est utilisée pour créer l'Interface Homme-Machine de notre projet. Elle est composée de plusieurs méthodes :

- public void start(Stage primaryStage) : permet de construire et démarrer l'application
- private VBox createParamGroup(String labelText, Control control) : permet de créer un groupe de deux paramètres dans une VBox
- private VBox createParamGroup(String labelText, Slider slider, Label valueLabel) : permet de créer un groupe de trois paramètres dans une VBox
- private VBox createImageBoxImportable(String defaultText, String caption) : permet de créer une VBox qui contient une image et un bouton
- private VBox createDynamicImageBoxNoised(String defaultText, String caption) : permet de créer une VBox qui contiendra l'image bruitée

- private VBox createDynamicImageBoxDenoised(String defaultText, String caption) : permet de créer une VBox qui contiendra l'image débruitée
- private VBox createMetricBox(double value, String label) : permet de créer une VBox qui contient les statistiques de l'image débruitée
- public static void main(String[ ] args) : appelle la méthode start qui lance l'application

2.3.5 Patch

Cette classe permet de crée des instances de patchs

- public Patch(double[ ] data, int x, int y) : un constructeur de patch depuis un vecteur
- public double[ ] toVector() : getter de l'attribut de Patch qui contient le vecteur qui représente le patch
- public static Patch fromVector(double[ ] v, int x, int y) : instancie un nouveau patch à partir d'un vecteurs et deux coordonnées

2.3.6 Tresholding

Cette classe contient l'ensemble des méthodes de seuillage utilisés sur les résultats de l'ACP

- public static double seuilVisu(double sigma, int size) : calcule le seuil utilisé par la méthode visu
- public static double seuilBayes(double sigma, double sigmaSignal) : calcule le seuil utilisé par la méthode Bayes
- public static double estimateGlobalSigmaSignal(double[ ][ ] contributions, double sigma) : estime l'écart-type du signal utile
- public static double soft(double lambda, double x) : applique un seuillage doux sur une variable
- public static double hard(double lambda, double x) : applique un seuillage dur sur une variable
- public static double[ ][ ] appliquerSeuillage(double[ ][ ] contributions, double lambda, boolean isSoft) : applique le suillage selon celui renseigné
- public static List⟨double[ ]⟩ reconstructionsDepuisContributions : reconsrtuits les données après le seuillage

2.3.7 Classes de stockage

Notre code contient également des classes de stockage :

- ACPResult : récupère les résultats de l'ACP
- ImageZone : récupère des sous-images extraites de l'image originale dans le but d'appliquer l'ACP locale
- MoyCovResult : récupère les résultats de la moyenne de covariance des vecteurs
- VectorWithPosition : permet de stocker un vecteur avec les coordonnées du patch qu'il représente

2.3.8 Main

Classe principale du programme permettant d'appliquer l'ensemble du processus de bruitage et de débruitage de l'image.

- public static void bruitage(String path, int sigma) throws Exception
  1. Charge une image
  2. Lui applique un bruit gaussien
  3. Enregistre l'image
- public static void debruitageGlobal(String pathOriginal, String pathNoisy, int sigma, int patchs, String extractionType, String seuillageMethod, String seuilType) throws Exception
  1. Extrait les patchs de l'image bruitée
  2. Convertit ce patch en vecteur
  3. Applique l'ACP sur ces vecteurs
  4. Projete les résultats
  5. Calcul le lambda pour le seuillage
  6. Applique le seuillage voulu
  7. Reconstruit l'image débruité
  8. Calcul les indicateurs de performances (MSE, PSNR, % d'amélioration)
  9. Sauvegarde les performances du débruitages dans un fichier texte
- public static void debruitageLocal(String pathOriginal, String pathNoisy, int sigma, int patchs, String extractionType, String seuillageMethod, String seuilType) throws Exception
- public static BufferedImage loadImage(String path) throws Exception : permet de charger une image depuis un chemin placé en paramètre
- public static void saveImage(BufferedImage img, String path) throws Exception : permet de sauvegarder une image dans un répertoire dont le chemin est placé en paramètre

2.4 Approche individuelle

Pour la répartition des tâches lors de ce projet, nous nous sommes appuyés sur les capacités de chacun des membres du groupe. Dans les grandes lignes, nous avons suivis cette répartition :

Antonin : code de l'application (hors IHM)

Clément : écriture des rendus et démonstration mathématiques, analyse UML

Mathias : code de l'application (hors IHM) et rapport intermédiaire

Mattias : écriture des rendus, commentaires du code, analyse UML

Thomas : code de l'IHM

Vous pourrez trouver une répartition des tâches plus précise sur GitHub (analyse/gantt/ganttL1.mmd) et un aperçu en annexe (6)

En plus de l'utilisation de GitHub pour l'organisation du code et des rendus, nous consacrions 15 minutes pour que chaque personne explique au reste du groupe ce qu'elle a fait et ce qu'elle projette de faire ou d'améliorer afin de progresser dans une même direction et de nous adapter aux travaux de chacun.

2.5 Choix initiaux

2.5.1 Diagramme de classe et de séquence

Pour le diagramme de classe, la structure générale n'a pas été modifié. Nous avons simplement ajouter des classes de stockage et ajouté certaines méthodes intermédiaires pour simplifier le code.

Pour le diagramme de séquences, nous avons écouté les retours de notre première réunion et modifié le diagramme afin qu'il corresponde à ce que nous avions réellement codé.

2.5.2 Répartition des tâches

Nous avons complété celle-ci afin qu'elle recouvre l'ensemble du projet. Comme expliquer précédemment, nous avons suivi la dynamique de la première phase avec 2 à 3 personnes concentrées sur le codes et les autres sur l'écriture des rapports ou de la documentation.

3 Présentation de notre interface utilisateur

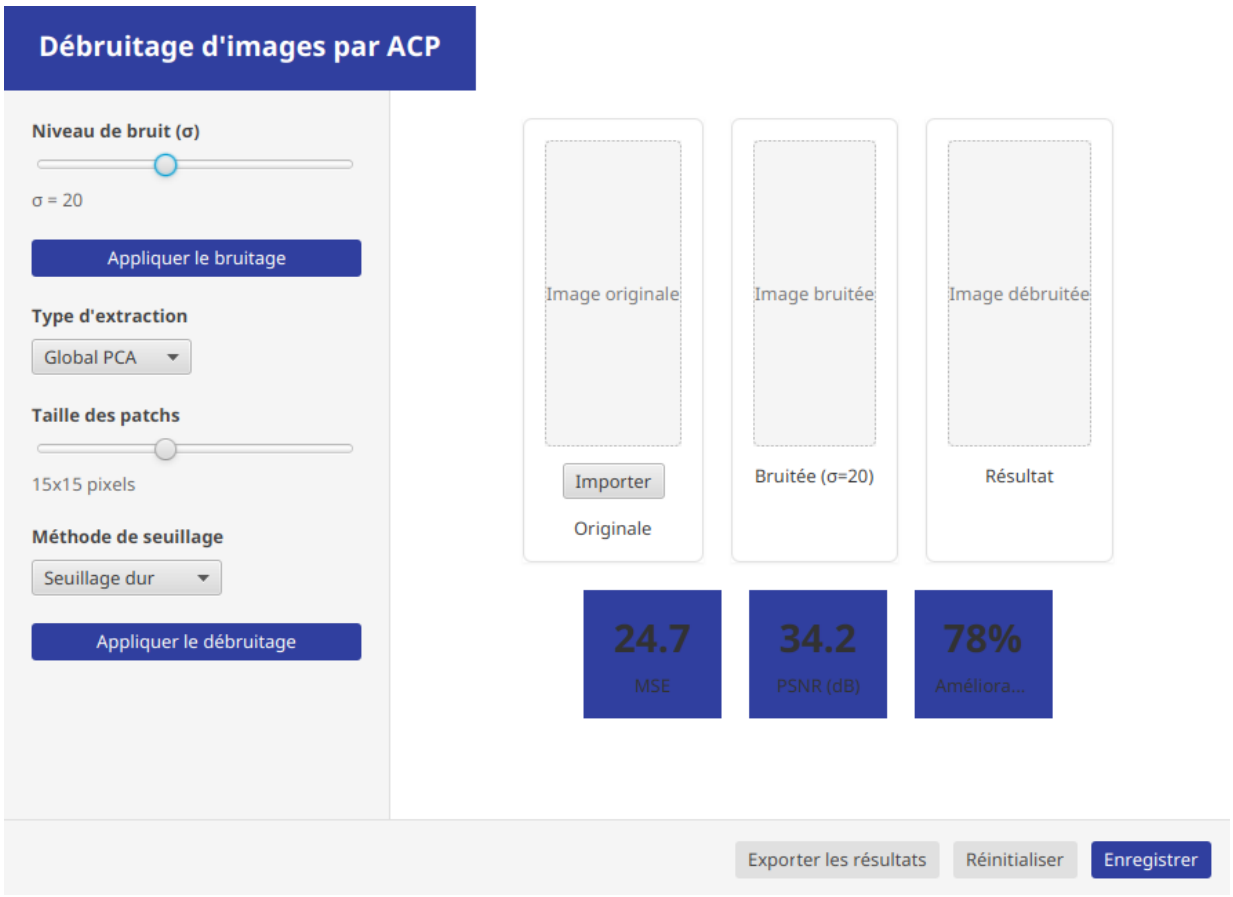

Figure 2: IHM de l'application

3.1 Présentation de l'interface

L'objectif de l'interface est de proposer les fonctionnalités de notre programme au travers d'une interface graphique permettant de sélectionner les différentes méthodes de seuillage, le type de seuil, la taille des patchs ainsi que la valeur du bruit.

Une autre fonctionnalité que nous voulions implémenter est l'affichage de l'image de départ, bruitée ainsi que le résultat final. Il est également important d'afficher les indicateurs de performances associés à la procédure que nous venons d'appliquer sur l'image.

Enfin, une action de débruitage optimale est implémentée dans l'IHM afin que les utilisateurs puissent, à partir d'une image bruitée, avoir le meilleur retour des huit méthodes.

3.2 Justification

Pour le principe de l'IHM, nous avons voulu suivre les étapes suivantes :

1. Charger une image localement
2. Lui appliquer un bruitage en fonction d'une constante σ choisis par l'utilisateur
3. Choisir le type d'extraction, la taille des patchs, la méthode de seuillage et le type de seuil
4. Lecture des indicateurs de performances
5. Exporter les résultats

Pour le choix du sigma ainsi que la taille des patchs, nous avons choisis des sliders permettant de sélectionner facilement une valeur dans un intervalle. Pour le reste des paramètres, nous avons utilisé des menus déroulants car le choix des possibilités était restreint.

Pour les actions de bruitage, débruitage sous paramètres et débruitage optimal, nous avons inséré trois boutons correspondant à chacune de ces actions.

Enfin, nous avons la possibilité d'enregistrer, de réinitialiser l'image au travers de deux boutons situés en bas de page

4 Présentation des résultats

4.1 Préambule

Dans tout ce projet, nous avions 3 indicateurs afin de déterminer la qualité de notre débruitage :

4.1.1 MSE

Il mesure la différence moyenne au carré entre l'image originale (de référence) et l'image débruitée.

$$\text{MSE } = \frac{1}{lc} \sum_{i=1}^{l} \sum_{j=1}^{c} (X(i,j) - Y(i,j))^2$$

avec X l'image bruité et Y l'image débruité, (l, c) le nombre de lignes et colonnes et (i, j) la position des pixels.

Pour son interprétation, un MSE= 0 signifie que les deux images sont identiques et plus il est élevé, plus l'image débruité est dégradée

4.1.2 PSNR

Le PSNR (en dB) compare le signal maximal possible (valeur maximale d'un pixel, dans notre cas 255) au bruit (erreur mesurée par le MSE).

$$\text{PSNR } = 10 \cdot \log_{10} \left(\frac{255^2}{\text{MSE}} \right)$$

Pour son interprétation, on suit les règles suivantes :

- inférieur à 20 dB : qualité médiocre
- entre 30 dB et 40 dB : qualité acceptable voire bonne
- supérieur à 40 dB : très bonne qualité

4.1.3 Amélioration

Il s'agit d'une variable que nous avons ajouté et qui permet de définir un pourcentage d'amélioration en suivant la formule suivante :

$$\text{amelioration } = \frac{\text{mseBruitee } - \text{ mse}}{\text{mseBruitee}} * 100$$

4.2 Valeur idéale de taille de patch

Dans un premier temps, nous avons cherché à déterminer quelle était la taille optimale de patch pour obtenir les meilleures performances de débruitage. La table ci-dessous présente les résultats obtenus avec une méthode globale, un seuillage dur et un seuil de type VisuShrink (sigma = 20, image Lena).

Taille du patch	MSE	PSNR	Amélioration
5x5	48.20	31.30	87.92 %
6x6	43.34	31.76	89.14 %
7x7	41.27	31.97	89.66 %
8x8	40.52	32.05	89.85 %
9x9	40.66	32.04	89.81 %
10x10	41.17	31.98	89.68 %
11x11	41.55	31.94	89.59 %
12x12	42.26	31.87	89.41 %

Table 1: Valeurs de performances selon la taille des patchs

On observe que la performance augmente avec la taille du patch jusqu'à 8x8, puis se stabilise, voire régresse légèrement. La taille 8x8 fournit le meilleur compromis, avec une MSE minimale (40.52) et un PSNR maximal (32.05 dB). Cette taille offre donc une couverture spatiale suffisante pour l'ACP tout en évitant de mélanger des zones trop hétérogènes.

4.3 Limites de notre programme

Nous avons ensuite évalué les limites de notre programme du point de vue des temps d'exécution, en faisant varier la taille de l'image d'entrée. Le tableau ci-dessous indique le temps total nécessaire pour effectuer les 8 méthodes de débruitage, pour un niveau de bruit de $\sigma = 25$.

Taille de l'image	128x128	256x256	512x512	1024x1024
Temps d'exécution	1,2 s	3,5 s	11,4 s	43,3 s

Table 2: Temps d'exécution selon la taille de l'image

On constate une croissance quasi-exponentielle du temps d'exécution avec la taille de l'image. Cette tendance est attendue, car une image plus grande implique un plus grand nombre de pixels, donc un plus grand nombre de patchs à extraire, projeter, seuiller et reconstruire. Cette complexité s'applique pour chacune des 8 méthodes, expliquant les temps cumulés observés.

4.4 Détermination de la meilleure méthode

Enfin, nous avons comparé les 8 combinaisons possibles de méthode de seuillage (doux ou dur), calcul de seuil (VisuShrink ou BayesShrink), et mode d'ACP (global ou local). Les résultats suivants ont été obtenus sur l'image Lena bruitée avec $\sigma = 25$, en utilisant des patchs 8x8.

Méthode utilisée	MSE	PSNR	Amélioration
Global/Doux/Visu	246.55	24.21	44.26 %
Global/Doux/Bayes	212.78	24.85	51.90 %
Global/Dur/Visu	87.89	28.69	80.13 %
Global/Dur/Bayes	421.16	21.89	4.79 %
Local/Doux/Visu	243.26	24.27	45.01 %
Local/Doux/Bayes	202.40	25.07	54.24 %
Local/Dur/Visu	87.04	28.73	80.32 %
Local/Dur/Bayes	416.45	21.94	5.85 %

Table 3: Performances des différentes méthodes

L'analyse des résultats montre clairement que les méthodes utilisant un seuillage dur couplé à VisuShrink (en global ou local) offrent les meilleurs scores en MSE et PSNR, avec environ 80% d'amélioration. La meilleure méthode est Local/Dur/Visu avec une MSE de 87.04 et un PSNR de 28.73 dB.

Les méthodes utilisant BayesShrink en combinaison avec un seuillage dur donnent au contraire les pires performances, ce qui s'explique par une estimation de seuil trop faible et une élimination excessive du signal utile.

Les variantes douces (soft) donnent des résultats intermédiaires, avec un meilleur équilibre entre atténuation du bruit et conservation des détails.

On retiendra donc que la meilleure méthode sur cette image est celle utilisant un seuillage dur avec VisuShrink, en ACP locale.

En annexe (3), vous trouverez un de nos résultats pour la méthode Global/Dur/Visu sur l'image Lena.

5 Conclusion

5.1 Résultats

Pour ce qui concerne les résultats, même si les indicateurs mathématiques nous disent que l'un de nos débruitage est très bon, l'image finale obtenue paraît bien, mais reste avec un léger flou. Les résultats obtenus avec notre batterie de méthodes sont satisfaisants mais loin d'être parfait.

On notera en particulier que les méthodes de Bayes ne donnent jamais de bon résultats, le bruit persiste dans les images débruitées.

Dans les recherches auxiliaires que nous avons faites, nous avons trouvé un article où des personnes ont développé une IA capable de supprimer un bruit gaussien sans connaître le σ de départ. Cette IA combine les estimations de débruitage avec des probabilités pour différents niveaux de bruit. Tout cela est appuyé par un système de neurones qui optimise les paramètres de débruitage.

Ces recherches permettent de mettre en perspective nos résultats et de les comprendre. Nous remarquons qu'il est nécessaire de déployer des gros moyens (IA, apprentissage) afin de réussir un débruitage sur des bruits gaussiens qui soit très bon. Cependant, il est rappelé dans l'article que ce système ne fonctionne que sur des bruits gaussien et non suivant un loi de Poisson ou autre, ce qui montre la difficulté de la suppression des bruits.

Pour revenir sur des solutions plus simple que nous pourrions implémenter, il existe une méthode de débruitage qui fusionne la méthode globale et local que nous avons utilisé. D'après certaines études du sujet, de plus gros programme prennent comme base de code une fusion entre une méthode global et local afin de conserver les détails fins avec le local et l'aperçu générale de l'image avec le global.

5.2 Expériences

Ce projet a été bénéfique pour nous tous et nous a permis de développer des capacités individuelles qui nous serons utiles pour le monde professionnel.

Pour les personnes concentrés sur le code, ils ont accrus leurs compétences en Java ainsi que dans le codages d'applications plus importantes que ce qui a été vu en TP. Ils devaient également se contraindre aux analyses UML faites en amont.

Pour tous les membres du groupe, nous avons dû apprendre à comprendre du code écrit par quelqu'un d'autre et de s'en servir.

Pour ce qui est des soft-skills, la communication, le travail en groupe et l'autonomie ont été travaillés durant ce projet.

5.3 Ouverture

Ce travail ouvre la voie à des améliorations concrètes : intégrer une fusion locale/globale, ou utiliser des librairies d'IA comme TensorFlow pour le pré-traitement. Dans un contexte professionnel, où l'analyse d'images médicales ou satellitaires exige à la fois précision et rapidité, de telles optimisations deviendraient indispensables.

6 Annexes

Figure 3: Lena (image source)

Figure 4: Lena (image bruité : sigma = 20)

Figure 5: Lena (image débruité (Global/Dur/Visu))

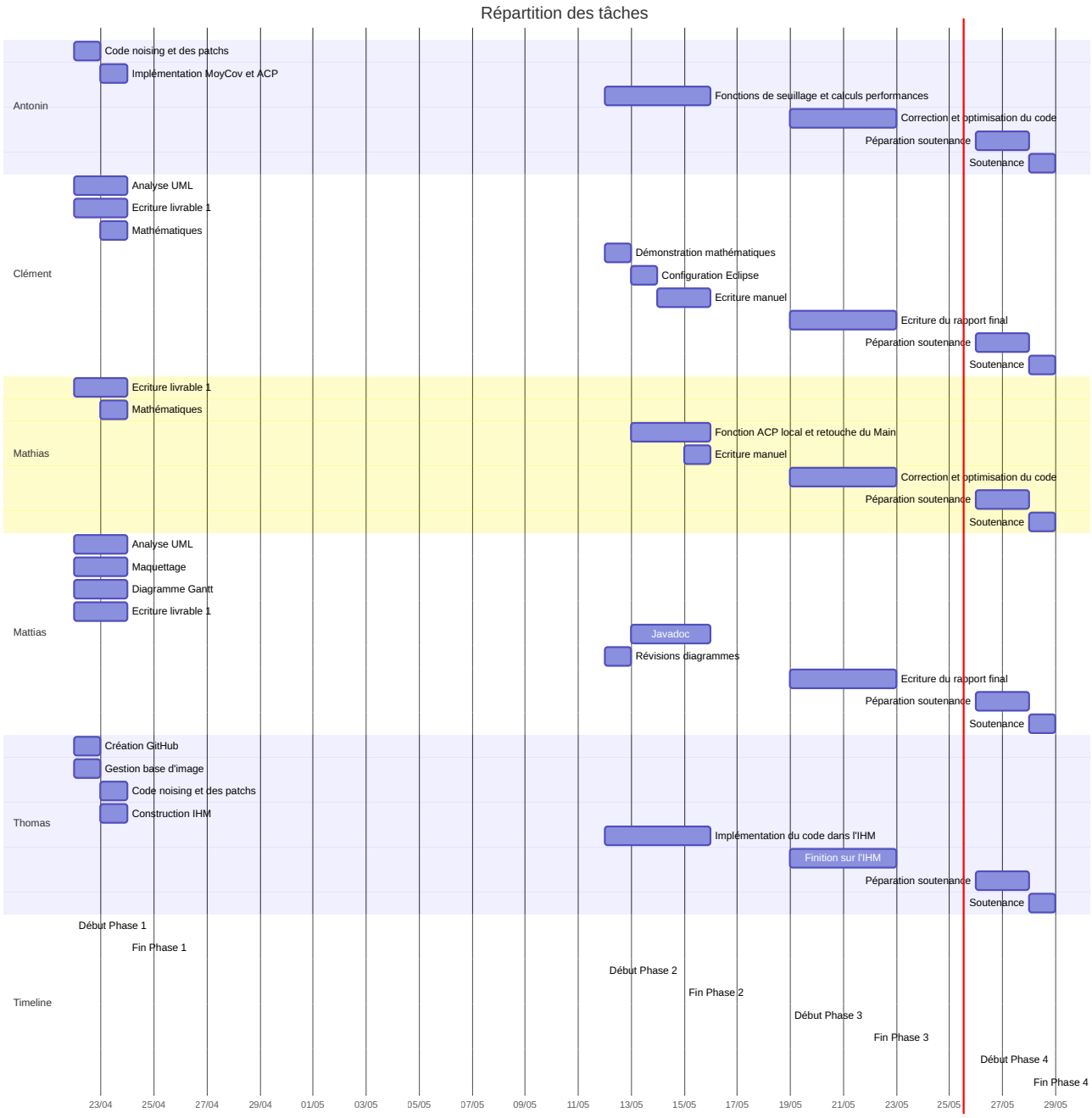

Figure 6: Diagramme de Gantt

Références

[1] Majed El Helou and Sabine Süsstrunk. "Blind Universal Bayesian Image Denoising With Gaussian Noise Level Learning". In: *IEEE Transactions on Image Processing* 29 (2020), pp. 4885–4897. ISSN: 1941-0042. URL: https://ieeexplore.ieee.org/abstract/document/9024220.

[2] Nisrine FORTIN CAMDAVANT. *Débruitage d'images par analyse en composantes principales*. Document interne fourni en préambule du projet. 2025.

[3] Nisrine FORTIN CAMDAVANT. *SAE - Descriptif et objectifs*. Document interne fourni en préambule du projet. 2025.

[4] Nisrine FORTIN CAMDAVANT and Elisabeth RANISAVLJEVIC. *Présentation de la SAE*. Document interne fourni en préambule du projet. 2025.

[5] Rini MAYASARI and Nono HERYANA. "Reduce Noise in Computed Tomography Image using Adaptive Gaussian Filter". In: *International Journal of Computer Techniques* 6.1 (2019), pp. 17–20. ISSN: 2394-2231. URL: https://arxiv.org/pdf/1902.05985.

[6] W. Wu et al. "SwinDenoising: A Local and Global Feature Fusion Algorithm for Infrared Image Denoising". In: *Mathematics* 12 (2024). URL: https://doi.org/10.3390/math12192968.