Rapport intermédiaire 1/2
Traitement des données audio-visuelles

SADURNI Thomas

Département Sciences du Numérique - Filière Image et Multimédia
2020-2021

Table des matières

Table des figures

1 Introduction

Ce document comprend un rapport intermédiaire des six premières séances de TP de Traitement des données audio-visuelles. Nous (oui, j'utiliserai le *nous* plutôt que le *je* tout au long du rapport) verrons ensemble les résultats des différentes séances de travaux pratiques et nous aborderons les six sujets suivants : l'estimation de paramètres, la régularisation, les champs de Markov, les processus ponctuels, la restauration d'images et enfin le photomontage. Nous essayerons de détailler au mieux les résultats obtenus et nous analyserons l'influence des paramètres sur ceux-ci. Les deux premiers TP explorent les modèles graphiques probabilistes, détaillés lors de cours par Vincent CHARVILLAT. Dans les quatre suivants, nous utiliserons les méthodes variationnelles vues en cours avec Yvain QUEAU, chercheur au CRNS à l'université de Caen, et mises en pratique dans ces TPs.
En général, la majorité du code des TPs était fournie, nous devions implanter les fonctions principales comprises à l'intérieur. L'objectif ici n'est pas de détailler le code mais de discuter des résultats. Chaque TP comprend des exercices obligatoires (entre 1 et 2) et un dernier facultatif.

2 Modèles graphiques probabilistes

2.1 Estimation de paramètres

Commençons par nous intéresser à l'estimation de paramètres dans le cas d'une courbe de Bézier à partir de données d'apprentissage. Les données d'apprentissage, bruitées ou non, sont l'ensemble des points d'abscisses comprises dans $[0, 1]$ et d'ordonnées calculées selon la formule suivante : $y_i = f(\beta_0, \beta, \beta_d, x_j) + b_i$ avec b_i le bruit suivant une loi gaussienne, si on veut les bruiter. On considère ici β_0 et β_d connus et appartenant à la courbe. L'objectif est de trouver les $\beta = [\beta_1, ..., \beta_{d-1}]$ "attracteurs" pour estimer la courbe de manière probabiliste à partir des données d'apprentissage, en résolvant un problème de minimisation.

Dans le premier exercice, nous devions faire une estimation du degré de la courbe d'apprentissage noté d. Au sens des moindres carrés et avec un degré $d = 2$, nous estimons les β et nous obtenons les résultats disponibles sur la figure 1. Ici, la courbe du modèle exact est seulement une référence pour le modèle estimé. En pratique, nous ne connaissons pas ce modèle et nous avons seulement les données d'apprentissage (croix sur la figure).

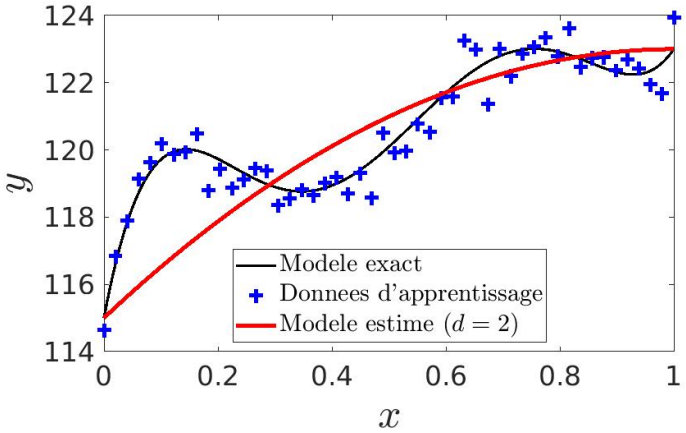

FIGURE 1 – Estimation de la courbe avec un degré d=2

On remarque très clairement que le degré que nous avons estimé est trop faible, le modèle estimé est bien loin du modèle exact. Le résultat est cohérent car on remarque rapidement que le modèle exact n'est clairement pas de degré 2. Ainsi, il vient naturellement d'augmenter d, voici plusieurs

courbes estimées avec des valeurs de d différentes.

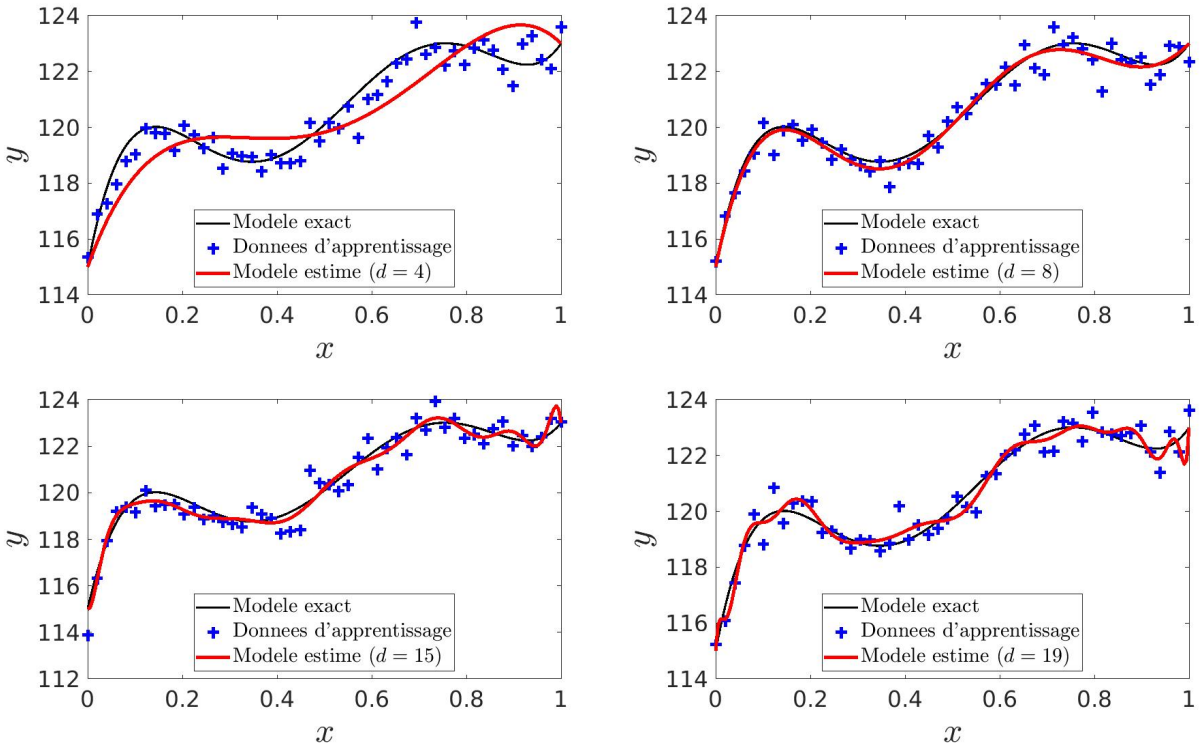

FIGURE 2 – Modèles estimés avec quatre valeurs de d différentes

Sur la figure 2, on remarque que plus d est grand, plus la courbe est proche du modèle exact. En revanche, si d est trop grand (par exemple pour $d = 15$ ou $d19$) on observe des variations aberrantes de la courbe. Il y a eu un sur-apprentissage des données et le modèle estimé est incorrect. Il faut donc réussir à savoir avec quel degré d le modèle estimé est le plus proche du modèle exact.

Pour cela, nous allons calculer l'erreur d'apprentissage, définie comme l'écart quadratique moyen entre les y_i des données d'apprentissage et les valeurs $y_{estimes}$ obtenues à partir du modèle appris, et l'erreur de généralisation définie de la même façon mais avec des données de test générées à partir des données d'apprentissage (attention, on ne prend pas en compte ces nouvelles données dans le calcul de β !).

On remarque immédiatement que les erreurs se stabilisent lorsque d vaut 5. Ainsi, on peut supposer que ce degré correspond au degré de la courbe du modèle exact. De plus on observe, lorsque d tend vers 20, que l'erreur d'apprentissage diminue et que l'erreur de généralisation augmente par rapport à la valeur pour $d = 5$. Ainsi, la valeur de d idéale pour se rapprocher au plus proche des données d'apprentissage est 5 voire 6, mais pas plus.

Si nous n'avons pas la possibilité de générer de données de test, on peut se contenter de conserver les données d'apprentissage en utilisant la "validation croisée" (VC de type $leave - one - out$). Cette méthode consiste à laisser une valeur des données d'apprentissage de côté, et à utiliser chaque point (x_i, y_i) $n-1$ fois pour l'apprentissage et une fois pour les tests, pour estimer le vecteur β_{estime}. Après avoir implanté la fonction avec la formule donnée dans le sujet du $TP1$, nous obtenons la courbe de la figure 6.

Encore une fois, pour des valeurs entre 3 et 5, la VC diminue fortement jusqu'à 5, ce qui confirme que notre modèle est au minimum de degré 5. De plus, lorsqu'on se rapproche de $d = 16$, la validation

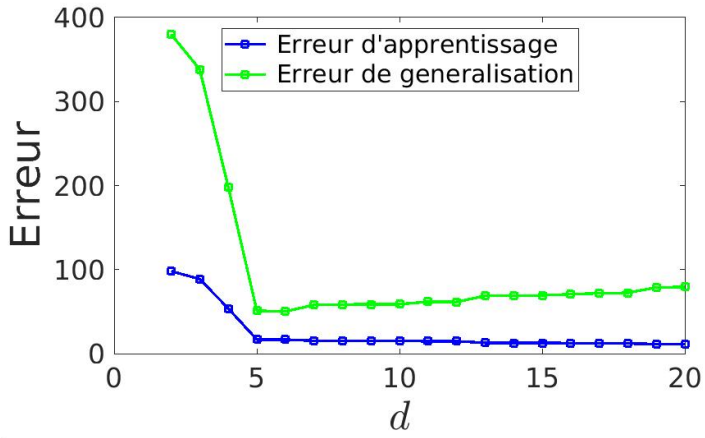

FIGURE 3 – Courbes des deux erreurs

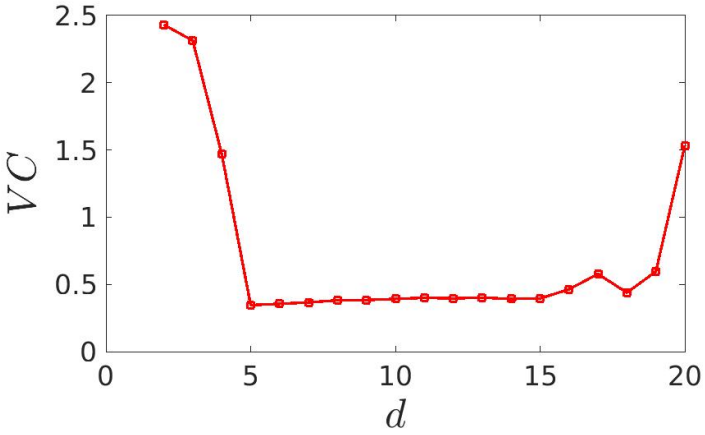

FIGURE 4 – Résultat de la validation croisée

croisée augmente et rejoint les résultats de la figure 2.

Dans ce premier TP, nous avons estimé les paramètres β dans le cas d'une courbe de Bézier pour en déduire son degré à partir de données d'apprentissage. Ici, d semble être 5 ou 6.

2.2 Régularisation

Reprenons le premier TP et intéressons-nous maintenant à des problèmes de plus grande complexité (d plus élevé). Pour cela, nous allons utiliser la régularisation en ajoutant un terme de pénalisation aux valeurs aberrantes de l'erreur d'apprentissage, noté λ et considéré comme un hyperparamètre. Ce paramètre λ a beaucoup d'influence sur la courbe estimée. Regardons plusieurs valeurs de λ sur le même problème que précédemment pour vérifier ces propos.

Avec une valeur de λ très élevée (environ 90), nous retrouvons la droite reliant les points aux extrémités des données (P_0P_1). Plus λ diminue et plus on s'approche du modèle exact, jusqu'à une certaine limite où, si la valeur est trop faible, nous faisons réapparaître le sur-apprentissage que nous avons observé précédemment :

Ainsi, il faut trouver un juste milieu pour choisir une valeur λ et obtenir une courbe correcte. Reprenons les fonctions de calcul de la validation croisée du premier TP et adaptons-les au problème actuel. Le but ici est d'estimer un hyperparamètre λ avec d fixé. Nous étions dans un problème de minimisation des paramètres inconnus β, et nous faisons de même ici en trouvant le minimum de VC en fonction de λ.

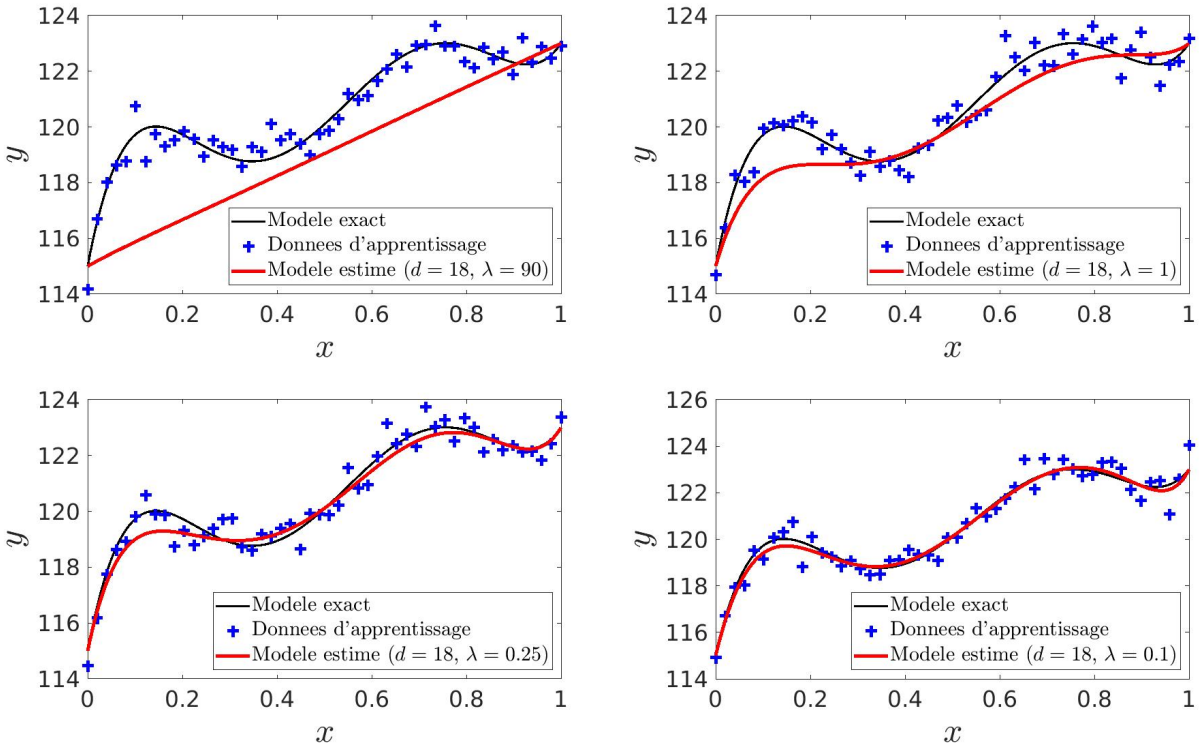

FIGURE 5 – Modèles estimés avec quatre valeurs de λ différentes et d fixé à 18

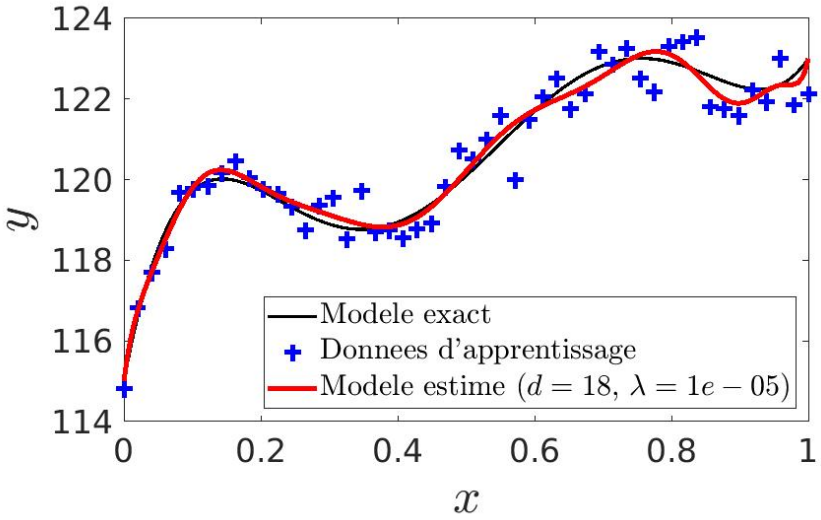

FIGURE 6 – Réapparition du sur-apprentissage si λ trop faible

On en conclut que la régularisation permet de contrôler la complexité du modèle en estimant un λ idéal. Appliquons maintenant cela à à la réalité augmentée avec une simulation de flamme de bougie.

2.3 Simulation d'une flamme de bougie

On souhaite simuler de la façon la plus réaliste possible une séquence d'images de flamme de bougie.

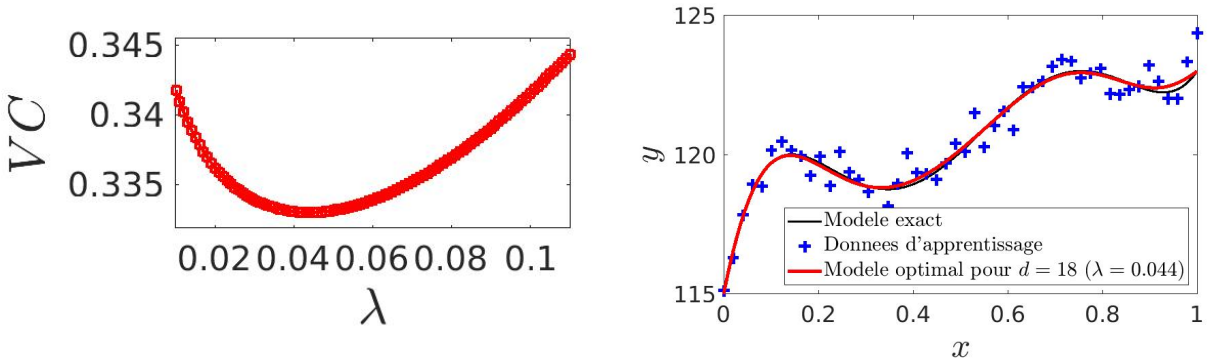

FIGURE 7 – Détermination du minimum de la validation croisée et courbe estimée avec l'hyper-paramètre associé

2.3.1 Estimation pour deux courbes de Bézier

Chaque partie de la bougie peut être modélisée par une courbe de Bézier de degré d, il y a donc deux listes de paramètres à estimer, β et γ. Mais pour que la flamme soit "fermée", il faut coupler ces deux listes, c'est-à-dire faire en sorte que le point de contrôle au sommet de la flamme d'ordonnée $y = 1$ soit commun aux deux courbes. La séquence d'image obtenue est donné à la figure 8.

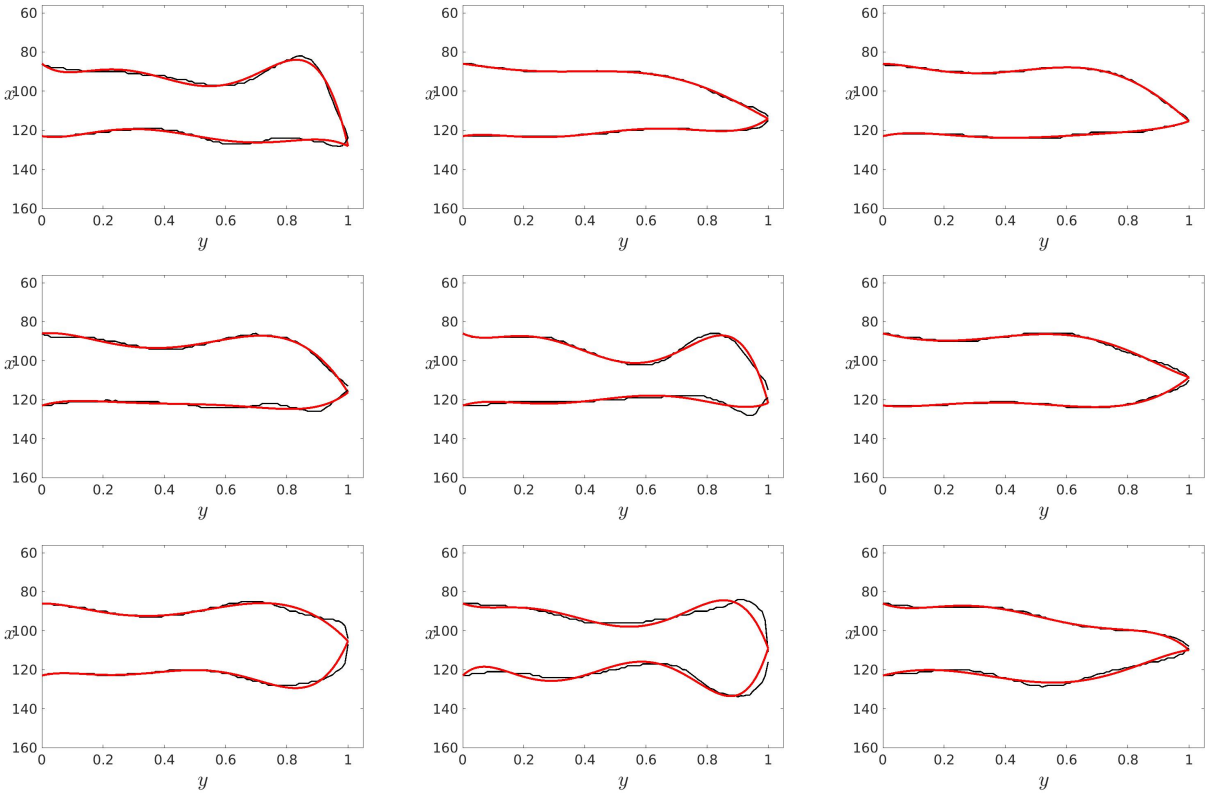

FIGURE 8 – Séquence d'image de l'estimation des flammes à l'aide de deux courbes de Bézier

Les deux courbes sont liées et la bougie est donc fermée. On remarque que l'estimation se déroule assez bien car nous reconnaissons une flamme de bougie (il faut cependant pencher la tête).

2.3.2 Estimation par tirages aléatoires

Pour estimer une flamme de bougie par tirages aléatoires, on modélise les distributions des abscisses des points de contrôle par des lois normales, puis on utilise les lois estimées pour procéder au tirage aléatoire de nouveaux points de contrôle. On implante une fonction qui permet d'estimer la moyenne et l'écart-type de chacun des points de contrôle, considéré comme une variable aléatoire, à partir des réalisations empiriques que constituent les données réelles et une autre, qui doit simuler une silhouette de flamme par tirage aléatoire de valeurs pour les paramètres β et γ. Voici les deux séquences d'images montrant les résultats, dont une en réalité augmentée (figure 10).

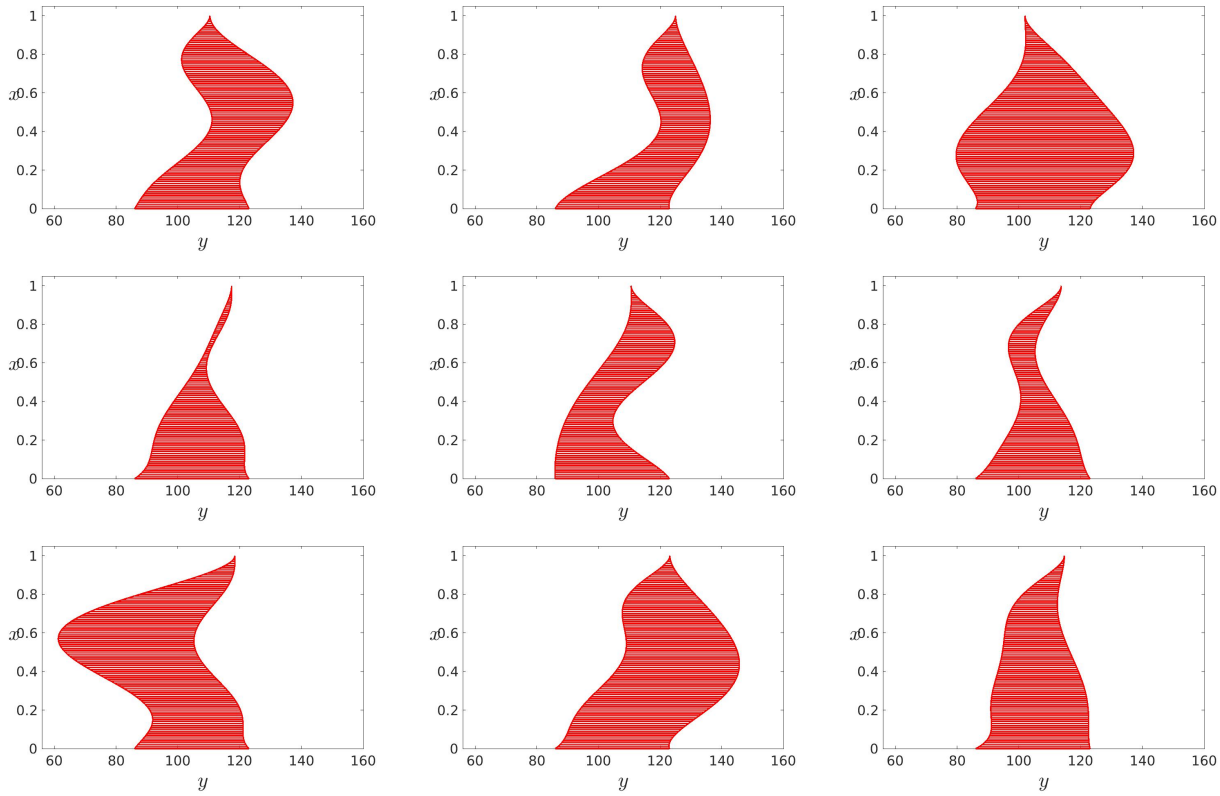

FIGURE 9 – Séquence d'image de l'estimation des flammes à l'aide de tirages aléatoires

Ainsi, l'estimation de paramètres par la méthode des courbes de Bézier, ou par tirages aléatoire peut être utilisé pour faire de la réalité augmentée comme nous l'avons montré au cours du deuxième TP. Lors de ces deux premiers TP, nous avons eu l'occasion de manipuler les modèles graphiques probabilistes avec les courbes de Bézier et les tirages aléatoires. A partir de données d'apprentissage, que nous pouvons augmenter avec des données tests ou en les réutilisant, nous pouvons maintenant établir le modèle mathématique associé à ces données.
Nous pourrions étendre nos études à la reconnaissance de formes (comme expliqué dans cette thèse) de mouvements (ici) ou la reconnaissance faciale comme vu en cours avec M. Charvillat.

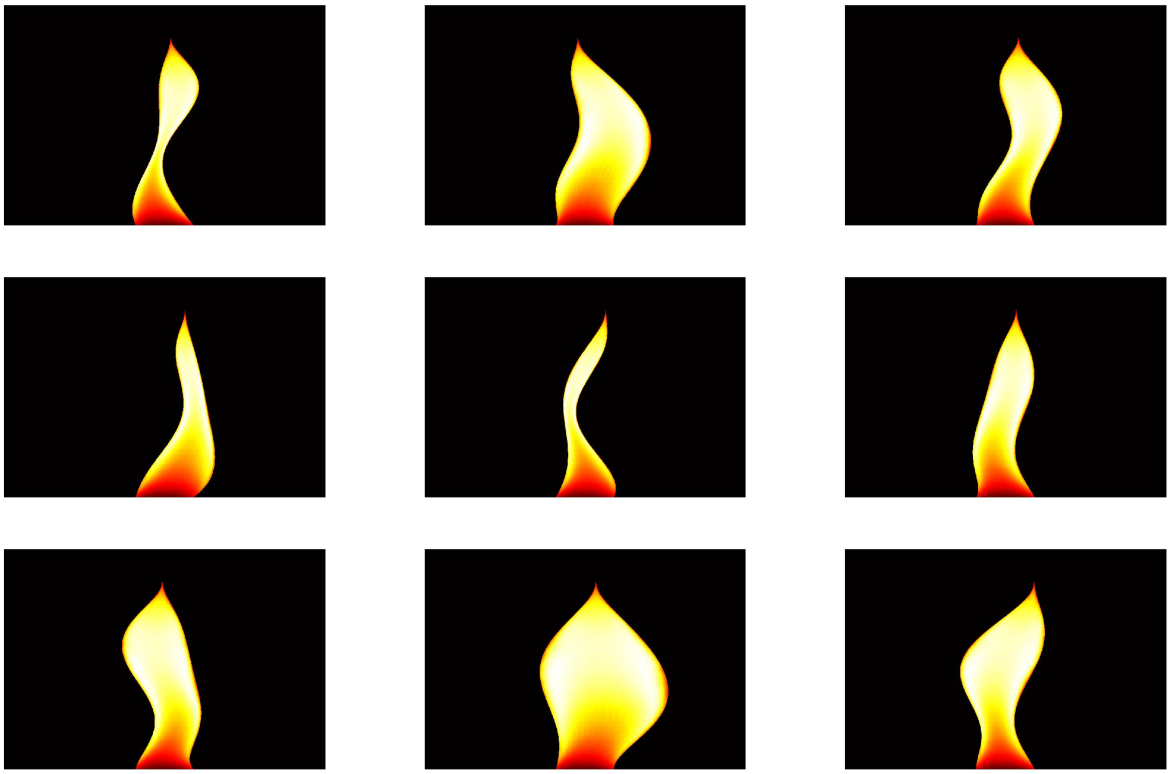

FIGURE 10 – Séquence d'image de l'estimation des flammes en réalité augmentée

3 Méthodes variationelles

Dans cette seconde partie, nous allons nous pencher sur les méthodes variationnelles utilisées notamment dans la vision par ordinateur. Les méthodes variationnelles permettent de résoudre les problèmes respectifs d'une manière mathématiquement transparente. Au lieu d'exécuter une séquence heuristique d'étapes de traitement, on commence par définir les propriétés qu'une solution doit avoir. Une fois que celles-ci sont fixées, l'algorithme approprié peut être dérivé "automatiquement". Les méthodes variationnelles sont particulièrement adaptées aux problèmes à dimensions infinies. Nous pouvons les retrouver dans de la segmentation d'images, dans le débruitage d'images, dans de l'*inpainting* (supprimer un objet de l'image), ou encore dans la compression d'images. A travers les différents TP, nous allons approfondir ces méthodes variationnelles à travers les domaines cités précédemment.

3.1 Segmentation par classification

La segmentation par classification en image est principalement utilisée pour séparer les objets du fond de l'image. Nous allons dans un premier temps, sur une image en niveau de gris, séparer en quatre classes différentes les formes de l'image donnée. Pour définir les quatre classes, l'utilisateur est demandé de sélectionner des régions correspondant aux différentes formes. Ainsi, une fois ces régions sélectionnées, le calcul du maximum de vraisemblance est lancé pour chaque pixel de l'image, et une classe lui est attribuée. Le résultat est donné dans la figure 11. On observe que certains pixels ne correspondent pas à la forme à laquelle ils devraient être attribués.

Pour remédier aux erreurs d'attributions, nous allons utiliser le *recuit simulé* qui consiste à parcourir tous les pixels de l'image, ligne par ligne et colonne par colonne, et comparer les deux énergies locales obtenues avec une classe k différente. On conserve l'énergie la plus faible et on lui attribue la classe k correspondante. Cet algorithme fait intervenir plusieurs paramètres comme la température T, le paramètre de régularisation β et α le paramètre de mise à jour de la température.

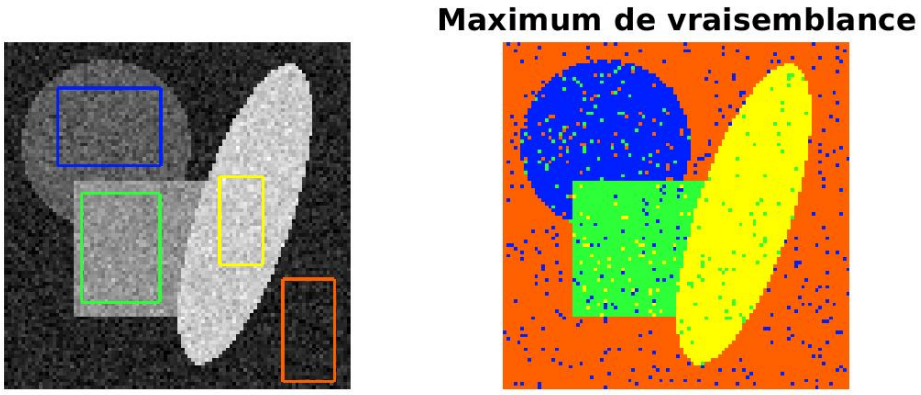

FIGURE 11 – Maximum de vraisemblance des pixels de l'image

En appliquant l'algorithme sur différentes valeurs de ces paramètres, nous obtenons des résultats proches de 100% de pixels correctement classés.

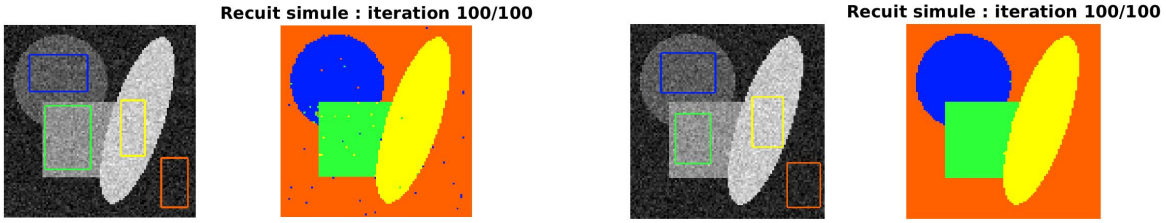

FIGURE 12 – Application de l'algorithme du recuit simulé sur une image en niveau de gris

L'image de droite de la figure 12 est juste à 99.4% avec les valeurs suivantes : $\beta = 3$ et $T = 2$. Sur l'image de gauche, $\beta = 3$ et il y a seulement 97% de bons pixels. Le terme de régularisation est donc un paramètre important à régler. En effet, le résultat en terme de pourcentage est mieux, mais les premières itérations de l'algorithme entraînent des modifications des classes de pixels incohérentes, pour ensuite converger vers une classe bien appropriée. Le paramètre T de la température joue un rôle important sur la convergence. On remarque que pour un β fixé à 3, la convergence vers 99% de bonnes attributions se fait en environ 60 itérations avec $T = 5$. En revanche, pour $T = 2$, seulement 20 itérations suffisent. Ces paramètres dépendent beaucoup de l'image utilisée, et nous devons faire plusieurs tests avant de trouver les valeurs optimales.

Que se passe-t-il si nous travaillons sur des images colorées RVB quelconques ? Il suffit simplement de changer quelques fonctions de notre algorithme et le tour est joué. Voyons ensemble le résultat sur différentes images. D'abord un magnifique paysage de la Guadeloupe, ensuite une supercar avec plus de 650 chevaux sous le capot et enfin une image sponsorisée par l'une des boissons les plus sucrées. Ici, nous avons évidemment fait plusieurs tests de paramètres en fonction de l'image et nous avons choisi de rester avec un nombre de classe fixé à 5. Libre à l'utilisateur d'essayer avec un nombre de classes et des paramètres différents (cf figure 13 et 14).

Il est clair que sur des images complexes comme celle que nous avons choisies, les résultats ne sont pas aussi bons que pour des images en noire et blanc avec des formes basiques. Néanmoins, la segmentation des images choisies se fait relativement bien et nous avons choisi seulement 5 classes pour segmenter ces images. Nous remarquons que les erreurs se font dans la plus grande partie des cas au niveau des ombres des objets. En effet, si on regarde sous notre belle berline de la figure 13, l'ombre est attribuée à la classe de la colline en arrière-plan, ce qui est logique car l'ombre est noire comme la colline. On raisonne de la même façon avec le capot de la voiture et le ciel : les deux sont des couleurs claires et peuvent être considérées comme les mêmes.

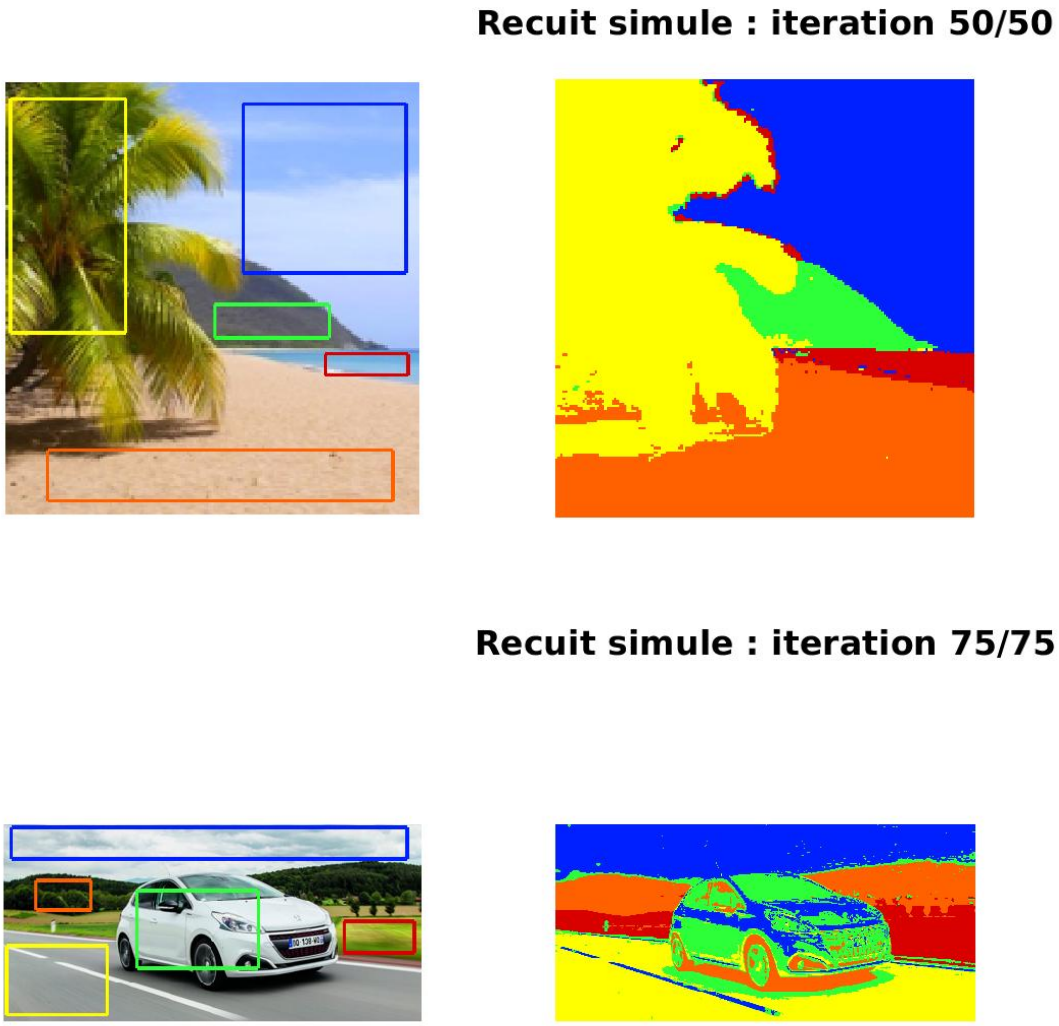

FIGURE 13 – Application de l'algorithme du recuit simulé sur des images en couleurs

Notons les effets des paramètres : si nous avions gardé $\alpha = 0.99$ les résultats n'auraient pas été aussi "bons", il y aurait eu des pixels parasites et une segmentation très imparfaite car l'image n'est pas simple à classifier. Ainsi pour ces images "complexes", nous fixons α à 0.9. Le paramètre T idéal semble, quant à lui, être autour de 2 et β 1 voire 2 pour l'ensemble des images. Notons aussi que le temps d'exécution est beaucoup plus lent pour les images complexes, il a fallu les compresser pour réduire leur taille et donc réduire le temps de quelques minutes (tout de même).

Ainsi, nous avons utilisé les méthodes variationnelles pour classifier les objets d'une image. Une autre manière de classifier des images est la segmentation par *kmeans* ou l'utilisation de l'intelligence artificielle par exemple. Dans notre cas, nous n'avons pas eu besoin d'utiliser le *machine learning*.

FIGURE 14 – Application de l'algorithme du recuit simulé sur le diabète

3.2 Détection d'objets dans l'image

L'objectif de ce TP est de détecter des objets dans une image, nous utiliserons une image aérienne de flamants roses. Cette technique permet de ne pas compter à la main les objets, qui peut être une tache longue et ennuyante s'ils ne se comptent pas sur les doigts de la main. Pour cela, un code naïf nous est fourni. Il consiste à positionner des disques de même rayon tirés selon une loi uniforme. Ensuite, pour chaque disque, on accepte la position si elle fait croître le niveau de gris moyen. Ainsi, la probabilité qu'un flamant rose se trouve à une position donnée est d'autant plus élevée que l'image est localement claire. Ce script naïf engendre cependant un problème : les disques étant indépendants, au bout d'un temps (ou d'itérations) infini(es), ils seront tous placés au même endroit (voir figure 15). Il apparaît donc nécessaire d'empêcher les disques de se chevaucher.

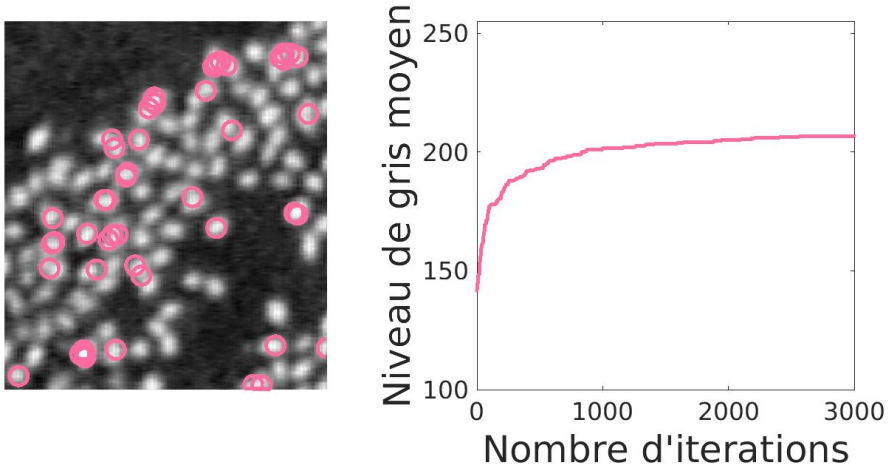

FIGURE 15 – Exécution de l'algorithme naïf de la détection de flamants roses

Pour pallier ce problème, nous allons appliquer les champs de Markov avec les N disques de l'image comme variables aléatoires du champ, en introduisant un a priori global sur une réalisation du champ de Markov qui pénalise les pairs de centres de disques trop proches. On impose donc une distance minimale entre les disques qui leur évite de se chevaucher.

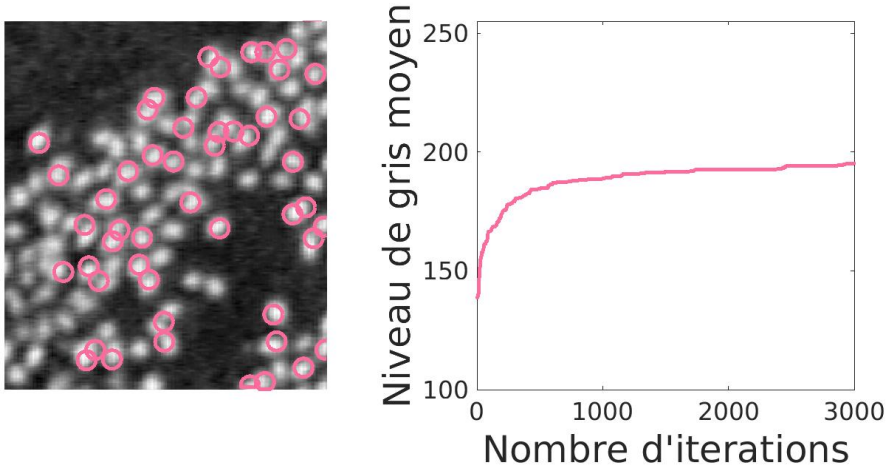

FIGURE 16 – Exécution de l'algorithme de détection de flamants roses par champs de Markov

Sur la figure 16, on observe que le niveau de gris à la limite est plus faible que sur la figure 15 dû au fait que les centres des disques ne tendent pas vers les mêmes coordonnées, plus claires. Le principal problème de cette méthode est qu'elle impose de donner, à priori, le nombre de flamants roses dans l'image, l'algorithme se contentant seulement de les placer correctement dans l'image. Pour améliorer davantage cet algorithme, nous allons utiliser le principe de *naissancemorts* pour minimiser l'énergie : les disques les moins pertinents au sens de l'énergie sont supprimés et des nouveaux disques sont ajoutés aléatoirement (suivant une loi de Poisson de moyenne λ) à la configuration courante, jusqu'à convergence, c'est-à-dire jusqu'à ce que le nombre de *morts* ne change plus.

On obtient le résultat sur la figure 17 et on compte environ 143 flamants roses. Cependant, il faut jouer sur plusieurs paramètres comme β pour donner plus ou moins d'importance au recouvrement des disques, S pour déterminer la taille à priori d'un flamant rose, γ pour l'intérêt donné à la couleur, T_0 qui prend en compte le nombre de naissances initiales, λ_0 qui correspond à la vitesse de variation du nombre de *naissances* et de *morts* et enfin α pour jouer sur la variation, au cours du temps, de la vitesse de changement du nombre de *naissances* et de *morts*.

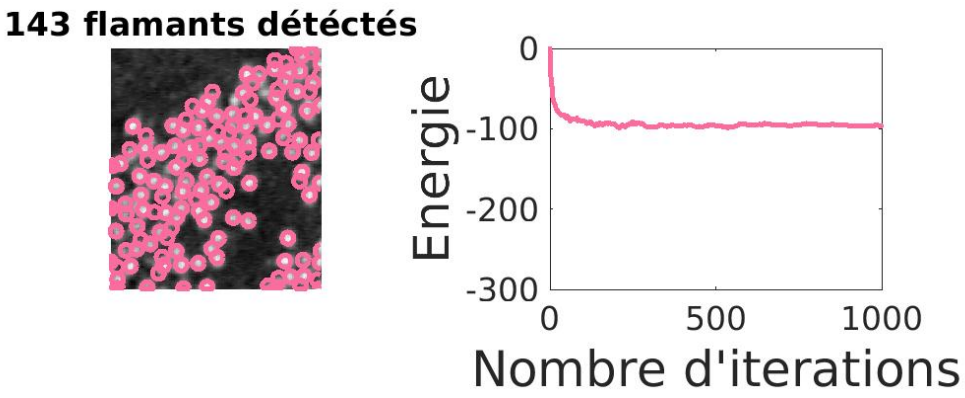

FIGURE 17 – Exécution de l'algorithme final de détection de flamants roses

3.3 Restauration d'images

Dans cette partie, nous nous attardons sur la restauration d'images. La restauration d'images est un problème inverse classique : étant donné une image observée et un modèle (généralement stochastique) d'un processus de dégradation de l'image, nous voulons restaurer l'image originale. Ce principe est notamment utilisé dans le débruitage et l'inpainting. Nous allons voir ces deux utilisations dans les paragraphes qui suivent.

3.3.1 Débruitage d'images

Le débruitage d'image est un exemple de restauration d'image où l'on suppose que la véritable image u est corrompue par un bruit (additif). Le débruitage de *Tikhonov* consiste à supprimer les *grains* d'une image, dans la majorité des cas ancienne. L'objectif est donc de lisser l'image initiale et obtenir une image finale qui soit assez proche. Ce modèle variationnel revient à minimiser l'énergie dite de *Tikhonov* et à résoudre l'équation d'Euler-Lagrange discrète présentées dans le sujet. L'exercice $naif$ donne le résultat suivant :

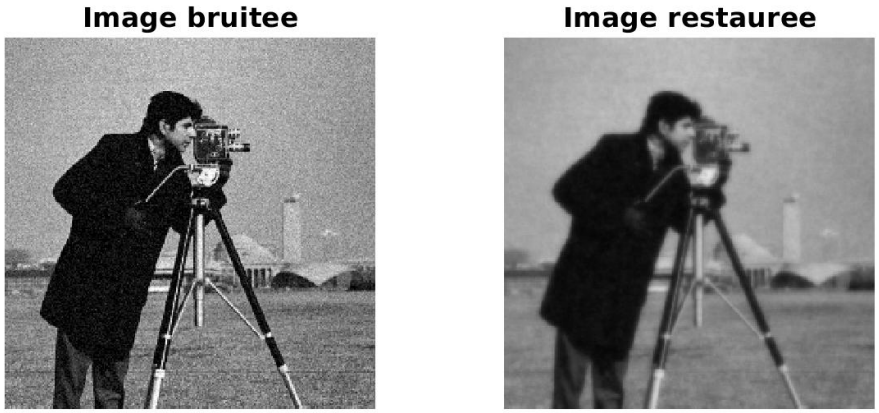

FIGURE 18 – Exécution de l'algorithme naïf de débruitage

L'influence du paramètre λ sur les résultats montre que plus il est grand, plus le bruit disparaît mais plus l'image est floutée, ce qui paraît normal car le terme du gradient de l'équation est prédominant dans ces conditions, ce qui floute l'image. Pour contourner ce problème, on choisit la variation totale en remplaçant le carré de la norme du gradient par la norme du gradient, pour rendre l'énergie non différentiable et on l'approxime par une fonction différentiable. Le résultat est donnée figure 19.

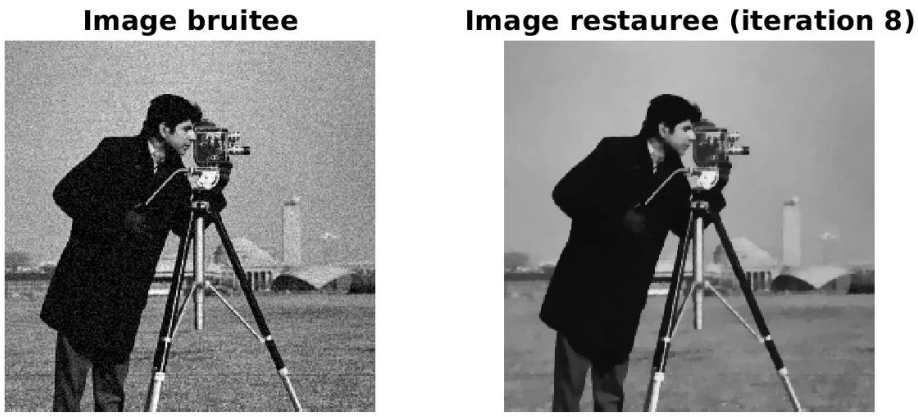

FIGURE 19 – Exécution de l'algorithme de débruitage par variation totale

On peut aussi appliquer cet algorithme sur une image en RVB, en séparant chaque canal de couleur pour obtenir une image moins bruitée figure 20.

FIGURE 20 – Exécution de l'algorithme de débruitage par variation totale sur une image RVB

3.3.2 Inpainting

L'inpainting d'image, ou retouche d'image en français, est une technique particulière de restauration d'image qui traite explicitement les données manquantes. Elle consiste à supprimer un texte ou un objet sur un certain domaine de l'image à partir de ce qui entoure ce domaine. Il existe deux cas de figure courants : la restauration de zones endommagées et la réalité diminuée. Nous allons voir ces deux cas à travers ce TP.

Pour utiliser la méthode par variation totale sur le modèle d'inpainting, le terme d'attache aux données ne doit être calculé partout en dehors du domaine endommagé. Dans un premier temps, nous avons un domaine D déjà donné dans une image des sources. Voici ce que nous donne l'image résultante :

FIGURE 21 – Inpainting sur une image endommagée

On remarque que la zone jaune a totalement disparu et a été remplacée par un domaine semblable à l'image non endommagée. On peut aussi utiliser la couleur comme critère pour déterminer le domaine. On remarque alors que, sur l'image avec la grenouille (figure 22), que le résultat est très correct. On peut ajouter une remarque : dès la première itération le résultat est très satisfaisant.

FIGURE 22 – Inpainting sur une image endommagée avec la couleur comme critère de sélection

On peut utiliser cette méthode pour essayer de gommer un randonneur d'une image par exemple, mais le résultat obtenu à la figure 23 parle de lui-même. Il faut recourir à une autre méthode d'inpainting : la réalité diminuée.

FIGURE 23 – Inpainting sur une image complexe de randonneurs

3.3.3 Réalité diminuée

Cette partie du TP fait l'objet des exercices facultatifs. Le but est de supprimer un ou plusieurs objets de la scène. La réalité diminuée ne découle pas d'une approche variationnelle comme l'inpainting par diffusion vue dans le cas précédent. La méthode est appelée le rapiéçage et consiste à remplacer les pixels manquants au voisinage par des pixels (de même position) appartenant au voisinage qui ressemble le plus. L'algorithme d'inpainting par rapiéçage est donné dans le sujet du TP. Appliqué à l'image des randonneurs, voici ce que nous obtenons :

FIGURE 24 – Inpainting par rapiécage

On remarque que les résultats de la figure 24 sont beaucoup plus réalistes que ceux obtenus sur la figure 23. Mais cette technique peut être encore amélioré en mettant en place un ordre de priorité sur les pixels à traiter.

La conclusion du TP nous mène à des publications d'amélioration du résultat et de logiciel faisant le travail d'une façon très réaliste (*Photoshop n'a cas bien se tenir !*)

3.4 Retouche d'images

Dans la suite des TP précédents, celui-ci montre comment les méthodes variationnelles peuvent être utilisées pour faire de la retouche d'image consistant à remplacer une partie d'une image *cible* par une autre image *source*. C'est le principe de photomontage par collage. Encore une fois il nous est donné un script naïf qui demande à l'utilisateur de sélectionner un polygone dans l'image source puis de le tracer dans l'image cible. On constate sur la figure 25 que le résultat est loin d'être réaliste, il n'y a aucun effort sur les contours, la chromatographie n'est pas la même sur les deux images. Le but de ce TP est de corriger ce problème.

FIGURE 25 – Photomontage par collage naïf, loin d'être considéré comme une image réelle

L'implantation des équations données dans le sujet et des indications (utiles) donne la figure 28. Le collage doit être effectué canal par canal et l'utilisateur peut choisir les images à retoucher qu'il souhaite. L'intérêt n'est pas de détourer très minutieusement l'objet que l'on veut incruster, sinon le programme ne sert à rien ... il faut avoir un polygone assez grossier et observer les résultats. On voit la différence sur le résultat en fonction du polygone choisi sur la figure 28

Le résultat est ici bien meilleur puisque la chromatographie semble la même sur les deux parties de l'image cible. En revanche, au niveau des éclaboussures de la baleine il est difficile pour l'algorithme de choisir si elles font partie de l'animal ou de de l'arrière-plan. De plus sur l'image du tableau de Van Gogh on remarque que les contours ne se sont pas bien $fondus$.

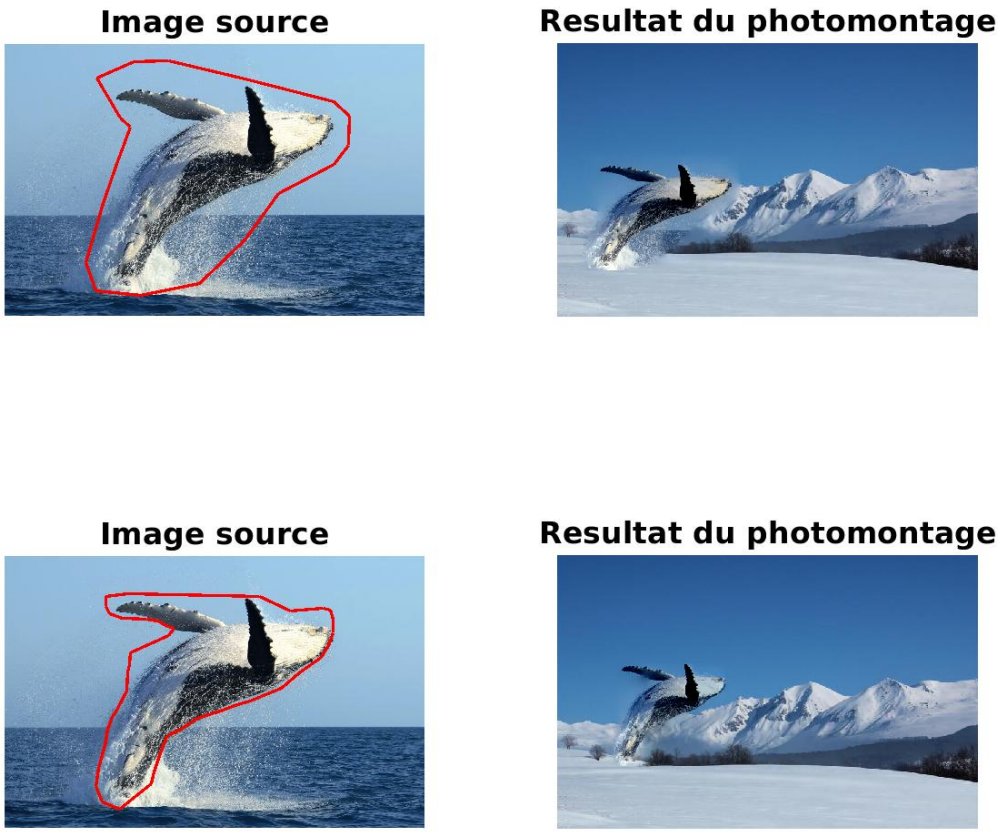

FIGURE 26 – Photomontages par collage de la baleine

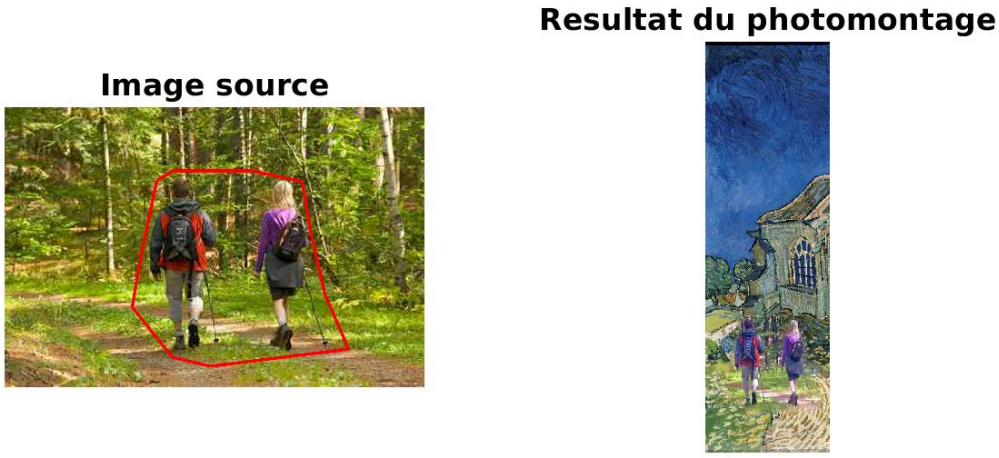

FIGURE 27 – Photomontage par collage des randonneurs sur le tableau de Van Gogh

Un autre avantage de l'inpainting est la décoloration d'image. On voit souvent sur les réseaux sociaux des images en noir et blanc avec seulement un objet en couleur, qui fait donc ressortir ce dernier pour en faire la promotion ou pour créer des images artistiques. Photoshop fait très bien ce travail mais il est plus intéressant de se pencher sur le côté mathématique de ce principe. Le sujet

nous explique en détail les bases mathématiques du problème. Ainsi, il est possible de décolorer une partie de l'image source et modifier le reste de l'objet en noir et blanc. Cela consiste à transformer une image en niveau de gris seulement sur un certain domaine, sélectionné par l'utilisateur. Là aussi, le but n'est pas de faire un contour très détailler mais de rester sur un polygone grossier pour observer les effets de la transformation. Celle-ci peut aussi être réalisable en utilisant le format LAB. Dans les images suivantes, on souhaite faire ressortir les pétales de la fleur et le camion rouge de la marque de boisson sucrée dont tout le monde raffole (pas moi).

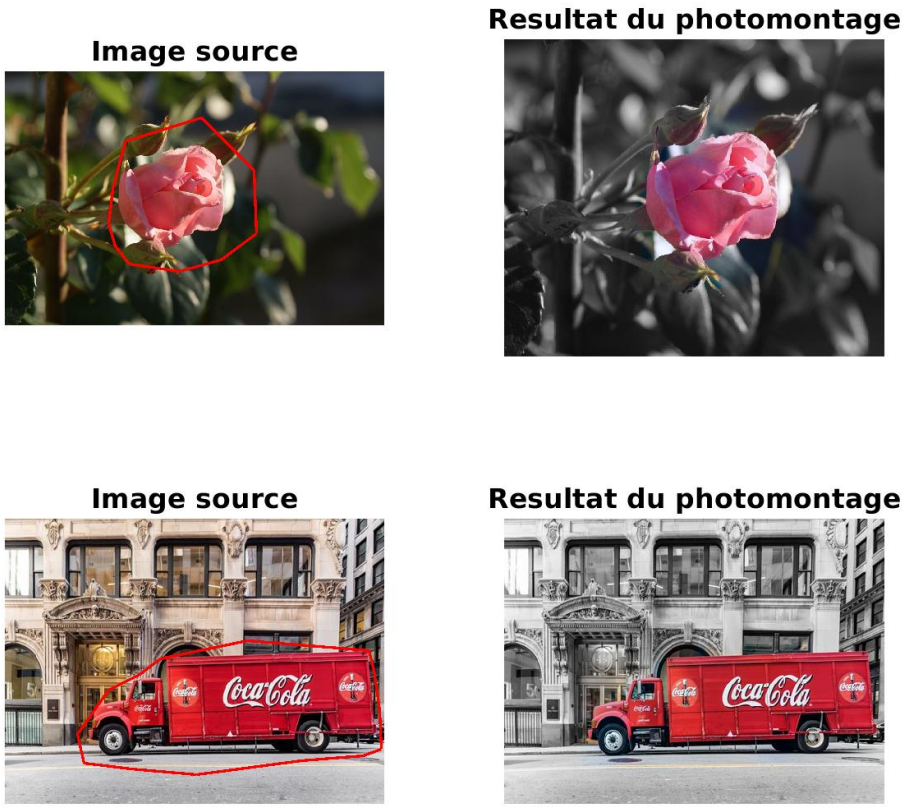

FIGURE 28 – Décoloration partielle d'une image

Pour conclure sur cette seconde partie de la première série de TPs, nous avons pu manipuler des images et faire ce pourquoi nous avons choisi cette filière : PHOTOSHOP !
Plus sérieusement, le côté mathématiques de la retouche d'image, de l'inpainting, du photomontage est très intéressant et nous voyant directement les effets des applications mathématiques que nous utilisons, ce qui n'est pas le cas dans la plupart des matières à l'ENSEEIHT. Nous avons une vision concrète des choses que nous pouvons faire en imagerie et je trouve cela très intéressant. Le côté ludique des cours et des explications lors des TPs nous permet de mieux comprendre comment *imaginer* et implanter les fonctions en *Matlab*. Je ne sais pas si j'ai obtenu les meilleurs résultats possibles pour les six TPs présentés ci-dessus mais je pense avoir des résultats corrects et rigoureux pour pouvoir comprendre, d'un point de vue extérieur, ce que j'ai l'occasion de faire en cours de *TAV*. Les TPs ne sont vraiment pas simples, et il faut, à mon sens, travailler beaucoup chez soi le week-end (ou la veille du rendu dépendant du profil de l'étudiant) pour avoir un résultat correct. Pour ma part, je consacrais environ trois heures en plus des séances pour essayer de finir le TP est avoir de bons résultats, ce qui est à mon sens, un peu trop.