N.B. Le règlement de TP stipule que chaque groupe d'étudiants doit sélectionner une image parmi celles fournies pour l'ensemble du TP, voir Moodle. Vous êtes libres de choisir l'image selon vos préférences. De plus, pour répondre aux questions, vous devrez appliquer les algorithmes d'optimisation appropriés pour traiter l'image choisie.

****

1 Introduction et manipulation des images

Une image en couleur peut être représentée par une variable u dans $\mathbb{R}^{m\times n\times c}$, où m et n représentent respectivement la hauteur et la largeur de l'image en pixels, et c est le nombre de canaux de couleur. Traditionnellement, pour les images courantes telles que les photographies grand public, $c = 3$, correspondant aux canaux Rouge (R), Vert (G) et Bleu (B). Cependant, il existe également une classe d'images en niveaux de gris, où $c = 1$, représentée par une matrice $\mathbb{R}^{m\times n}$. Les valeurs des coefficients de u

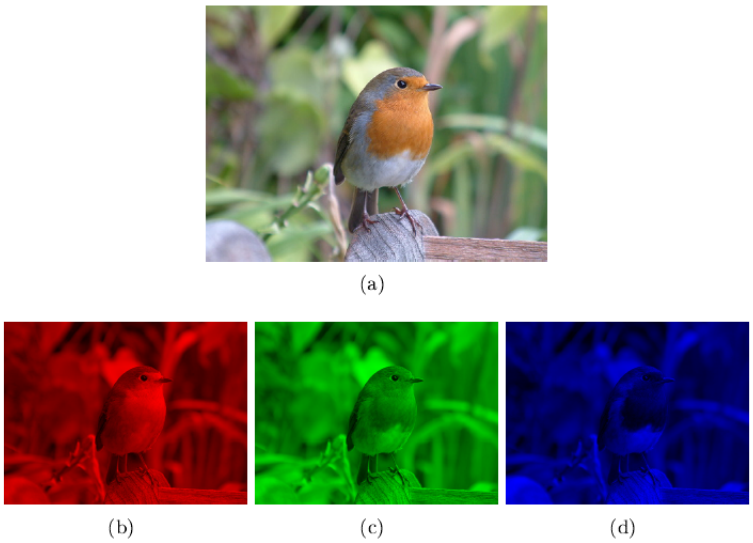

FIGURE 1 – (a) Exemple d'images en couleur. (b) Canal rouge. (b) Canal vert. (c) Canal bleu. Le canal bleu apparaît très sombre car cette composante est peu présente dans l'image (a). On voit a contrario qu'au niveau de la face du rouge-gorge, les plumes sont plus claires dans le canal rouge que dans le canal vert, puisqu'elles sont orangées, tandis que le fond, plutôt vert, apparaît plus clair dans le canal vert que dans le canal rouge. Ces images ont étés tirées des "cours du Pr. J.F Aujol et de Pr. T. Pham"

déterminent la dynamique de l'image. Par exemple, pour les images codées sur 8 bits(formats jpeg, ou png), ces valeurs sont des entiers compris entre 0 et 255, offrant ainsi 256 possibilités de valeurs. Dans le cas des images en niveaux de gris, une valeur proche de 0 correspond à une teinte sombre (proche du noir), tandis qu'une valeur proche de 255 correspond à une teinte claire (proche du blanc). Fondamentalement, ces valeurs sont liées au nombre de photons captés par le capteur associé à chaque pixel. Une pratique courante consiste à normaliser les valeurs de u pour qu'elles appartiennent à l'intervalle continu

$[0, 1]$. Pour ce faire, il suffit de diviser chaque valeur de u par 255 dans le cas des images codées sur 8 bits.

Dans la suite, nous nous concentrons spécifiquement sur les images en niveaux de gris afin de simplifier les calculs. Pour convertir une image, couleur ne niveaux de gris, nous pouvons utiliser plusieurs techniques en fonction des besoins spécifiques et des préférences en termes de résultats visuels et de précision dans la représentation des niveaux de gris : la moyenne des valeurs de trois canaux de couleur, la luminosité maximale/minimale des trois couleurs, la de-saturation, ...

Ici, nous vous suggérons de travailler avec une image selon votre choix pour explorer ces techniques de traitement et manipulation d'images en répondant aux questions ci-dessous :

1. Charger l'image choisie dans votre environnement du travail en utilisant une bibliothèque de traitement d'image : PIL (Python Imaging Library), OpenCV, Pillow, ... Afficher l'image pour s'assurer de bonne lecture. Si vous utilisez : **imread** récupère des images de **uint8** et pour quelques opérations dans le projet il est nécessaire de convertir en **double**.
2. Convertir l'image couleur en niveaux de gris en utilisant la méthode de la moyenne des trois canaux de couleur (rouge, vert, bleu). Afficher l'image obtenue.
3. Ecrire une fonction python permettant de réaliser les tâches suivantes : re-dimensionner l'image à une taille de 100×100 pixels, de faire pivoter l'image et la re-cadrer définir une région de 50×50 pixels centrée.
4. Écrire une fonction **modifiyImage**(Im, new_mean, new_std) qui renvoie une image new_I qui est une version de l"image Im de moyenne new_mean et d'écart-type new_std. Tester cette fonction pour augmenter le contraste de l'image "".
5. Calculer et afficher l'histogramme de l'image choisie sur 5 puis 256 intervalles à l'aide de la fonction **hist**.

Comme indiqué au début des travaux pratiques, nous souhaitons restaurer une image bruitée en utilisant les techniques d'optimisation numériques. Pour se faire, nous avons besoin de définir les notions ci-dessous :

— Gradient : un champs de vecteur $\nabla u \in \mathbb{R}^N \times \mathbb{R}^N$ utile en traitement d'images qui se caractérise à l'aide de différences finies. On a ainsi pour $u \in \mathbb{R}^N$ une image discrète de $N = m \times n$ pixels :

$$\nabla u[i, j] = \Big(\partial_x u_[i, j], \partial_y u_[i, j],\Big)$$

dont les dérivées partielles $\partial_x u_[i, j]$ et $\partial_y u_[i, j]$ sont définies, pour tout $(i, j) \in [[1; m]] \times [[1; n]]$, par

$$\partial_x u[i, j] = \begin{cases} u_{i+1,j} - u_{i,j} & si\, i < m, \\ 0 & sii = m \end{cases} \tag{1}$$

$$\partial_y u[i, j] = \begin{cases} u_{i,j+1} - u_{i,j} & si\, j < n \\ 0 & sij = n \end{cases}$$

1. En utilisant cette définition, écrire une fonction Python **grad.py** prend en argument une image et renvoie le gradient de cette image.
2. Écrire une fonction qui prend une image u en argument et renvoie une image N, telle que $N(i, j) = ||\nabla u[i, j]||$.

3. Charger votre image choisie et tester cette fonction en affichant la norme du gradient de cette image.
4. Que pouvons-nous dire des contours de l'image construite en choisissant une valeur "seuil" et affichant les points pour lesquels la norme du gradient excède cette valeur ?

— Divergence d'un champ du vecteur $p = (p_1, p_2) \in \mathbb{R}^N \times \mathbb{R}^N$: un scalaire div, mesurant le défaut de conservation du volume sous l'action du flot de ce champ p,est donnée par $\mathrm{div}p = \partial_x^* p^1 + \partial_y^* p^2$ telle que

$$\partial_x^* p^1[i,j] = \begin{cases} p^1_{i,j} & si\, i = 1 \\ p^1_{i,j} - p^1_{i-1,j} & si\, 1 < i < m, \\ -p^1_{i-1,j} & si\, i = m \end{cases}$$

$$\partial_y^* p^2[i,j] = \begin{cases} p^2_{i,j} & si\, j = 1 \\ p^2_{i,j} - p^2_{i,j-1} & si\, 1 < j < n \\ -p^2_{i,j-1} & si\, j = n \end{cases}$$

1. Écrire une fonction Python **div** qui prend en argument un champ de vecteurs et renvoie sa divergence.
2. Afin de vérifier que les opérateurs ∇ et div sont bien implémentés, il faut vérifier que pour tout u, ω, on a $< \nabla u, \omega >= - < u, \mathrm{div}\omega >$ en simulant aléatoirement u et ω.
3. Tester cette fonction pour une image en affichant le laplacien de cette image. Commenter le résultat.

2 Débruitage d'une image numérique

Entre l'acquisition d'une scène à l'aide d'un appareil photo et la génération du fichier numérique correspondant, divers phénomènes peuvent altérer l'image obtenue. Ces altérations sont communément appelées bruit. Le bruit devient particulièrement perceptible lors de la prise de photos en conditions de faible luminosité. En fait, le bruit est toujours présent, mais dans des scènes sombres, la luminosité des points que l'on souhaite capturer est faible, et de même ordre de grandeur que le bruit, ce qui rend ce dernier plus visible.

L'opération visant à réduire le bruit d'une image, ou à restaurer sa qualité originale, est appelée débruitage. Pour cela, il est essentiel de modéliser le phénomène de bruit. Un modèle simple et couramment utilisé est celui du bruit blanc gaussien. Dans ce modèle, on suppose que l'image capturée $v \in \mathbb{R}^N$ (donnée connue discrète de $N = m \times n$ pixels d'un problème mathématique ou physique, d'une image, des données statistiques,...) est la somme d'une image idéale $u \in \mathbb{R}^N$ (l'image que l'on cherche qui peut s'écrire aussi sous la forme $u = Ax$ avec A la matrice de dégradation) et d'un bruit blanc gaussien $b \in \mathbb{R}^N$:

$$v = u + b$$

Dans ce contexte, le bruit est considéré comme indépendant de la couleur des points de l'image, indépendant en chaque point de l'image, et suit une distribution gaussienne identique partout.

Afin de débruiter une image en niveaux de gris, nous adoptons le long de ce TP une approche basée sur les méthodes variationnelles. Ces méthodes consistent à définir une fonctionnelle J qui évalue la

similitude entre une image et le modèle de formation construit. Ainsi, le débruitage devient un problème d'optimisation. Si v représente l'image bruitée en niveaux de gris de taille $m \times n$, et si le bruit est un bruit additif gaussien, nous examinons une classe de fonctionnelles de la forme :

$$J(u) = \frac{1}{2}||v-u||_2^2 + \lambda R(u)$$

où $u \in \mathbb{R}^{m\times n}$ désigne l'image débruitée que nous cherchons à obtenir, $||\cdot||_2$ représente la norme euclidienne L^2 et $R(u) : \mathbb{R}^{m\times n} \longrightarrow \mathbb{R}^+$. Cette fonctionnelle fait intervenir deux termes essentiels pour la reconstruction d'image : le premier est appelé terme d'attache aux données, ou terme de fidélité qui force la reconstruction (idéalement, l'image u) à rester proche de l'image bruitée v. Il y a donc de multiples façons de le définir. Ici, on choisit la norme euclidienne $||\cdot||_2$ au carré à l'image bruitée v telle que l'erreur quadratique moyenne. En effet il a été démontré que ce choix est adapté au modèle choisi pour le bruit. Le deuxième terme R est appelé terme de régularisation qui traduit le modèle choisi pour l'image et impose une certaine structure à l'image débruitée. Dans ce projet, Cependant, le paramètre de pondération $\lambda > 0$ contrôle et régularise l'importance relative entre la fidélité aux données et la régularité de l'image débruitée.

Dans ce TP, nous nous intéressons à plusieurs fonctionnelle de R sous la forme $R(u) = r(\nabla_h u)$ où le gradient est donné par l'équation de différence finie (1) et nous cherchons l'image u qui minimise la fonctionnelle $J(u)$, exprimée par :

$$\min_{u\in\mathbb{R}^{m\times n}} J(u) = \frac{1}{2}||u-v||_2^2 + \lambda r(\nabla_h u).$$

Ce processus d'optimisation peut être réalisé au moyen de diverses techniques, telles que la descente de gradient, les méthodes de Newton, ou d'autres algorithmes d'optimisation numérique. L'objectif ultime est d'obtenir une image débruitée qui préserve les détails essentiels tout en réduisant le bruit indésirable

2.1 Débruitage d'image par la régularisation de Thikonov

On suppose que la différence de valeurs entre deux pixels voisins est faible dans l'image que l'on souhaite restaurer v. Autrement dit, que les variations d'intensité (de niveaux de gris) dans une image sont douces. Pour cela, on pose

$$r(\nabla u) = \frac{1}{2}||\nabla u||_2^2$$

Nous définissons donc un terme de régularisation qui prendra une valeur d'autant plus élevée que l'image u présente des variations brutales d'intensité (des pairs de pixels voisins ayant des valeurs très différentes). Ce terme mesure ainsi bien une distance au modèle choisi. Ici, les images v telles que $r(\nabla u)$ soit le plus faible sont les images constantes ne présentant aucune variation d'intensité. Nous nous intéressons finalement à trouver l'image u solution du problème d'optimisation suivant

$$J(u) = \frac{1}{2}||v-u||_2^2 + \frac{\lambda}{2}||\nabla u||_2^2$$

dont J est une fonction convexe, fortement convexe de module λ et différentiable telle que

$$\nabla J(u) = (u-v) - \lambda \mathrm{div}(\partial_x u, \partial_y u).$$

Ce problème d'optimisation peut s'écrie sous forme d'une équation d'Euler-Lagrange sous la forme $-(u(x,t) - v(x) - \lambda\Delta u) = 0$. Cette dernière n'admet pas de solution explicite simple et nous décidons

ainsi de définir une séquence minimisante en introduisant une variabilité temporelle de la descente de gradient :

$$\frac{\partial u}{\partial t}(x,t) = -\left(u(x,t) - v(x) - \lambda \Delta u\right)$$

que nous pouvons discrétiser par les différences finies comme suit :

$$\frac{u_{i,j}^{n+1} - u_{i,j}^{n}}{\Delta t} = -\left(u_{i,j}^{n} - v_{i,j} - \lambda \Delta u_{i,j}^{n}\right),$$
$$u_{i,j}^{n+1} = u_{i,j}^{n} - \Delta t\left(u_{i,j}^{n} - v_{i,j} - \lambda \Delta u_{i,j}^{n}\right).$$

Pour les expériences numériques, nous allons importer une image visuellement satisfaisante, puis ajouter un bruit blanc gaussien.

Remarque : il est possible que l'image importée comporte déjà du bruit, de sorte que la considérer comme le signal idéal u n'est pas très rigoureux. Cependant, on va ici rajouter un bruit très important (visible), de sorte que cette hypothèse restera raisonnable. Il faut également veiller à importer des images de taille suffisamment petite, pour éviter de manipuler des objets trop grands.

1. Utiliser **numpy.random.normal** pour ajouter un bruit Gaussien de déviation standard 10 à l'image initiale $v := u + \mathcal{N}(0; 10)$.
2. Adapter la méthode du gradient conjugué, gradient à pas optimal et la méthode BFGS pour résoudre le problème d'optimisation en choisissant $\lambda = 0.1$ et un nombre d'itérations de $iter_{\max} = 10^2$ et en initialisant $u_0 = g$ (g une image quelconque, vous pouvez prendre $g = v$). On note que le critère d'arrêt pourra être donné par le fait que l'image u ne bouge plus suffisamment.)
3. Calculer et sauvegarder les valeurs successives de la fonction objective $J(u)$. Tracer l'évolution de la fonction objective $J(u)$ en vérifiant qu'elle décroît bien à chaque instant et qu'elle converge.
4. Discuter de l'influence du paramètre λ sur la solution $\hat{u}$.
5. Tracer l'évolution de l'erreur quadratique moyenne entre l'image originale v et l'image estimée $\hat{u}_\lambda$ en fonction de λ. Commenter les résultats.
6. Comparer les résultats en implémentant l'évolution par l'équation d'Euler-Lagrange qui minimise $J(u)$ pour les mêmes données.

2.2 Débruitage pour le modèle de Rudin-Osher-Fatemi

Il s'agit ici d'un modèle complexe, mais plus proche de la réalité, introduit par Leonid Rudin, Stanley Osher, et Emad Fatemi en 1992 pour le débruitage d'image. Ce modèle est fondé sur l'idée de la variation totale (Total Variation, TV) et repose sur la minimisation d'une fonctionnelle d'énergie pour obtenir une version lissée de l'image tout en préservant les contours et les détails significatifs. En effet, les images dites naturelles présentent des caractéristiques intéressantes : pour simplifier, elles comportent des zones avec des variations très douces (voire pas de variations du tout "des aplats") délimitées par des bords francs. Les auteurs ont démontré qu'une meilleure régularisation consiste à remplacer la norme quadratique par la norme euclidienne :

$$r(\nabla u) = ||\nabla u||_2.$$

Nous sommes ainsi cette fois amené à résoudre le problème d'optimisation ci-dessous :

$$\min_{u \in \mathbb{R}^{m \times n}} J(u) = \frac{1}{2}||v - u||_2^2 + \lambda ||\nabla u||_2 \tag{2}$$

Il est clair que la fonction J n'est pas différentiable à cause du terme $||\nabla u||_2$. En particulier, il n'est plus possible d'utiliser la méthode du gradient dans ce cas. Pour remédier à ce problème, nous vous proposons d'utiliser deux approches : la première consiste à résoudre directement le problème d'optimisation (2) en introduisant des algorithmes plus complexe tels que : algorithme de Chambolle-Pock, algorithme d'accélération de Nestrov, algorithme de Polak-Rebera, algorithme de Barzilai ?Borwein, ... la deuxième approche consiste à modifier la fonction J en remplaçant le terme non différentiable par une variante approchée différentiable. Dans ce TP, nous nous intéressons qu'à la deuxième approche

en remplaçant le terme $||\nabla u||_2$ par une variante approchée différentiable sous forme d'une fonction d'activation

$$\phi_\alpha := \begin{cases} \mathbb{R} & \longrightarrow \mathbb{R}^+ \\ s & \longmapsto |s| - \alpha \ln\left(\frac{\alpha+|s|}{\alpha}\right) \end{cases} \qquad \alpha > 0.$$

Afin d'utiliser ϕ_α comme fonction d'activation, il suffit de répondre aux questions suivantes :

1. Monter que ϕ_α est de classe C^2 pour $\alpha > 0$. Calculer ϕ'_α et $\phi"_\alpha$.
2. En déduire que ϕ'_α est une fonction lipschitzienne de constante $k \leqslant \frac{1}{\alpha}$.
3. Vérifier que ϕ_α est convexe.
4. Écrire une fonction python **computePhi(s, alpha)** qui calcule trace les fonctions ϕ_α et $|\phi_\alpha|$ pour les valeurs de $\alpha \in \{0.25; 0.5; 1; 1.25; 1.5; 2\}$. Vérifier que la fonction approche de manière différentiable $|\phi_\alpha|$, l'approximation étant d'autant meilleure que α est petit.

En utilisant cette approximation, nous sommes donc amené à résoudre le problème d'optimisation :

$$\min_{u\in\mathbb{R}^{m\times n}} \tilde{J}(u) = \frac{1}{2}||v-u||_2^2 + \lambda \sum_{j=1}^{n}\sum_{i=1}^{n} \phi_\alpha\left(\partial_x u[i,j]\right) + \phi_\alpha\left(\partial_y u[i,j]\right). \tag{3}$$

On note que la solution image dans ce cas n'est pas assurée d'être proche de la solution image du problème d'optimisation (2), en effet l'approximation d'un terme dans les problème d'optimisation ne préserve pas les optimiseurs (minimiseurs ou maximiseurs).

Il est facile de prouver que la fonction $\tilde{J}$ est fortement convexe, différentiable et que, pour tout $u \in \mathbb{R}^{m\times n}$:

$$\nabla \tilde{J}(u) = (u-v) - \lambda \mathrm{div}\left(\phi'_\alpha\left(\partial_x u[i,j]\right); \phi'_\alpha\left(\partial_y u[i,j]\right)\right).$$

Pour les données suivantes : $\lambda = 4$, $u_0 = v$, un nombre total d'itérations de $iter_{\max} = 10^2$ et $\alpha \in \{10^{-2}; 10^{-1}\}$, résoudre le problème d'optimisation (3) en implémentant

1. La méthode de gradient à pas fixe et les méthodes DFP et BFGS.
2. Tracer l'évolution de l'erreur quadratique moyenne entre l'image originale v et l'image estimée $\hat{u}_\lambda$ en fonction de λ. Commenter.
3. Représenter la solution obtenue pour chaque méthode pour $\alpha = 10^{-2}; 10^{-1}$.
4. Commenter les résultats en comparant les images obtenues avec l'image initial v.

3 Inpainting ou désocclusion

Quand la valeur de certains pixels d'une image (solution des problèmes d'optimisation) est manquante (ou si nous souhaitons effacer un objet ou un artefact de l'image, l'inpainting vise à attribuer une valeur

à ces pixels de manière à ce qu'ils soient en harmonie avec le reste de l'image. Nous pouvons donc modéliser ce problème de la manière suivante :

$$\min_{u\in\mathbb{R}^{m\times n}} J(u) = \frac{1}{2}||\mathcal{R}u - v||_2^2 + \lambda \sum_{j=1}^{n}\sum_{i=1}^{n} \phi_\alpha\left(\partial_x u[i,j]\right) + \phi_\alpha\left(\partial_y u[i,j]\right).$$

où $\mathcal{R}$ est un opérateur de masquage qui supprime les pixels de la région inadmissible notée D_{mask} de l'image initiale u :

$$\mathcal{R}(u)(s) = \begin{cases} 0 & si\, s \in D_{mask} \\ u(s) & sinon \end{cases}$$

Nous pourrons implémenter l'équation d'Euler-Lagrange associée au problème d'optimisation :

$$\frac{\partial u}{\partial t} = 2\mathcal{R}^* v - 2\mathcal{R}^*\mathcal{R}u + \lambda \mathrm{div}\left(\frac{\phi_\alpha\left(||\nabla u||\right)}{||\nabla u||}\nabla u\right)$$

où $\mathcal{R}^*$ est l'opérateur adjoint (inverse) qui sert à garantir que les valeurs des pixels restaurés sont cohérentes avec les valeurs connues de l'image originale tout en permettant la reconstruction des pixels manquants de manière harmonieuse avec le reste de l'image.

Cependant, si l'on suppose que l'image v est exempte de bruit, nous ne souhaitons pas altérer les valeurs des pixels déjà connus. Nous aimerions donc que la condition suivante soit satisfaite :

$$||\mathcal{R}u - v||_2 = 0.$$

Autrement dit, cela signifie que $Ru(x) = v(x)$ en dehors des zones masquées, car nous ne voulons pas modifier l'image dans ces régions. Pour y parvenir, nous pourrions théoriquement laisser tendre λ vers 0, mais cela ralentirait considérablement la convergence de l'algorithme.

Pour accélérer le processus, nous adoptons l'algorithme en effectuant à chaque itération les étapes suivantes :

— Effectuer un pas dans la direction opposée au gradient du seul critère de régularité.

— Re-projetter sur les données en fixant les valeurs connues de $u(mask == 0) = v(mask == 0)$.

Cette approche garantit que les pixels connus restent inchangés tout en permettant l'optimisation efficace des pixels manquants, en suivant les étapes ci-dessous :

1. Pour tester votre programme, vous pouvez soit charger une image, créer un masque et appliquer votre algorithme, soit tester directement l'algorithme sur l'image générée par votre modèle.
2. Résolvez le problème d'optimisation en utilisant la méthode de gradient à pas fixe, la méthode de Newton et la méthode BFGS, en vous basant sur les mêmes données que dans la section précédente. Comparez et commentez les résultats obtenus avec chaque méthode.
3. Implémentez l'équation d'Euler-Lagrange en utilisant les définitions de divergence et de gradient que vous avez définies précédemment.