TP1 – Filtrage d'images par diffusion

Soit $I \in \mathbb{R}^{n_r \times n_c}$ une image en niveaux de gris, représentée par une matrice à n_r lignes et n_c colonnes. Le gradient $\nabla I = [I_x, I_y]$ de I peut être approché, en tout pixel $(x, y) \in [1, \ldots, n_c] \times [1, \ldots n_r]$, par différences finies avant selon les directions horizontales et verticales :

$$I_x(x,y) = \begin{cases} I(x+1,y) - I(x,y) & \text{si } x < n_c, \\ 0 & \text{si } x = n_c, \end{cases} \tag{1}$$

$$I_y(x,y) = \begin{cases} I(x,y+1) - I(x,y) & \text{si } y < n_r, \\ 0 & \text{si } y = n_r. \end{cases} \tag{2}$$

Le but de ce premier TP est de vous familiariser avec la manipulation d'images sous Matlab en implémentant ces opérations de dérivation par multiplication matricielle, puis en les utilisant pour réaliser le filtrage d'une image par diffusion.

Soit $n = n_r\, n_c$ le nombre de pixels de l'image, et $u \in \mathbb{R}^n$ le vecteur obtenu par concaténation des colonnes de I (en Matlab : `u = I(:);`). Alors il existe des matrices $D_x \in \mathbb{R}^{n \times n}$ et $D_y \in \mathbb{R}^{n \times n}$ telles que $u_x = D_x u$ et $u_y = D_y u$ soient les deux vecteur de $\mathbb{R}^n$ obtenus par concaténation des colonnes des matrices I_x et I_y définies ci-dessus.

On rappelle en outre que le laplacien est défini comme la divergence du gradient, et que la divergence étant l'opposé de l'adjoint du gradient : $\Delta I = -\nabla^\top \nabla I$. La matrice $L = -D_x^\top D_x - D_y^\top D_y$ est donc telle que $u_\Delta = L u_0$ soit le vecteur de $\mathbb{R}^n$ obtenu par concaténation des colonnes de l'image I_Δ du laplacien ΔI de I.

Exercice 1 : matrices de différences finies

Écrivez la fonction `finite_differences_2D`, permettant de calculer les matrices de différences finies D_x, D_y et L définies ci-dessus, pour une grille 2D de taille quelconque $n_r \times n_c$. Le script `exercice_1` vous permettra de vérifier l'exactitude du résultat sur une grille de taille 4×3, et de mesurer le temps de calcul pour des tailles croissantes de grilles.

Attention : la résolution des appareils photographiques numériques étant de plus en plus importante, l'efficacité de ces opérations simples est cruciale. Assurez-vous donc que le temps de calcul pour une résolution 4K (3840×2160) reste de l'ordre de quelques secondes. Il vous est conseillé pour cela d'implémenter les opérateurs D_x et D_y comme des matrices creuses (voir l'aide de la fonction `spdiags`).

Exercice 2 : gradient et laplacien d'une image

Écrivez la fonction `finite_differences_2D`, permettant de calculer les images I_x, I_y et I_Δ représentant les deux composantes du gradient de I et le laplacien de I, grâce aux opérateurs de différences finies construits dans l'exercice 1. Il vous faut pour cela :

1. concaténer les colonnes de la matrice I en un vecteur u ;
2. appliquer à gauche l'opérateur de différences finies approprié pour obtenir les vecteurs u_x, u_y et u_Δ ;
3. "déconcaténer" ces vecteurs pour former les images I_x, I_y et I_Δ.

Cette fonction renverra également l'image I_g de la norme du gradient de I, définie en tout point (x, y) par $I_g(x,y) = \|I_x(x,y), I_y(x,y)\| = \sqrt{I_x(x,y)^2 + I_y(x,y)^2}$.

Le script `exercice_2` vous permettra de vérifier l'exactitute des résultats par comparaison aux opérateurs prédéfinis dans Matlab `imgradientxy`, `imgradient` et `del2`, qui réalisent les mêmes opérations.

Attention : par défaut les opérations en Matlab sont *matricielles*. Par exemple, `A*B` est égal aux produits des matrices `A` et `B` ; les opérations *terme à terme* doivent être spécifiées par un point, par exemple `A.*B` pour le produit terme à terme des matrices `A` et `B`.

Exercice 3 : diffusion linéaire

Le filtrage gaussien d'une image $I_0 : \Omega \subset \mathbb{R}^2 \to \mathbb{R}$ peut être interprété comme un processus de diffusion linéaire, homogène et isotrope du niveau de gris de chaque pixel vers ses voisins, et représenté par l'équation aux dérivées partielles :

$$\begin{array}{rl} I^{t=0} = I_0 & \text{sur } \Omega, \\ \partial_t I^t = \Delta I^t & \text{sur } \Omega, \forall t > 0. \end{array} \tag{3}$$

On peut approcher la différentiation temporelle par $\partial_t I(x,y) = \frac{I^{t+1}(x,y) - I^t(x,y)}{\delta_t}$, pour un certain pas de temps $\delta_t > 0$, et l'opérateur laplacien Δ par des différences finies telles que celles des exercices précédents. En utilisant les mêmes notations que précédents, ceci conduit à la discrétisation suivante d'un pas de diffusion linéaire explicite :

$$u^{t+1} = u^t + \delta_t L u^t. \tag{4}$$

En utilisant les opérations de vectorisation, d'approximation du laplacien et de déconcaténation, écrivez la fonction `linear_diffusion_step` permettant de réaliser un pas de diffusion linéaire sur une image. Le script `exercice_3` vous permettra de visaliser l'évolution de l'image au cours du temps, ainsi que la valeur de $\frac{\|Lu^t\|}{\|Lu^0\|}$, qui doit être égal à zéro à la convergence, c'est-à-dire pour $t = +\infty$. Arrêter la diffusion après un temps fixé permet d'obtenir une approximation plus ou moins lisse de l'image initiale.

Le processus peut être accéléré en augmentant la valeur de δ_t, mais vous constaterez que le processus diverge si ce pas devient trop important. Pour parer à cela, on peut considérer la discrétisation implicite :

$$u^{t+1} = u^t + \delta_t L u^{t+1}. \tag{5}$$

de telle sorte que le vecteur $u^t + 1$ est solution du système linéaire $(Id - \delta_t L)\, u^{t+1} = u^t$. Modifiez la fonction `linear_diffusion_step` de manière à implémenter cette nouvelle discrétisation, et observez l'importance de ce choix en augmentant la valeur de δ_t jusqu'à obtenir une approximation quasi-uniforme de l'image initiale.

Exercice 4 : diffusion non linéaire

La diffusion linéaire permet de lisser une image, et donc d'en retirer le bruit, mais au prix de la perte des structures saillantes (les "contours"). L'idée de la diffusion non linéaire consiste à stopper le processus de diffusion le long de ces structures saillantes. On introduit pour cela un coefficient de diffusivité inversement proportionnel au gradient de l'image :

$$g(x,y) = \exp\left\{ -\frac{\|\nabla I(x,y)\|^2}{\lambda^2} \right\}, \qquad \forall (x,y) \in \Omega, \tag{6}$$

et on modifie l'équation de diffusion de la façon suivante :

$$\begin{array}{rl} I^{t=0} = I_0 & \text{sur } \Omega, \\ \partial_t I^t = \operatorname{div}\left(g^t \nabla I^t\right) & \text{sur } \Omega, \forall t > 0, \end{array} \tag{7}$$

(pour $g \equiv 1$, on retrouve bien sûr l'équation de diffusion linéaire, homogène et isotrope).

Cette équation peut être discrétisée comme précédemment, en introduisant une matrice diagonale $G \in \mathbb{R}^{n \times n}$ contenant les valeurs du coefficient de diffusivité g, et en se rappelant que la divergence est l'opposé de l'adjoint du gradient :

$$u^{t+1} = u^t + \delta_t(-D_x^\top G D_x - D_y^\top G D_y) u^{t+1}. \tag{8}$$

Écrivez la fonction `peronamalik_diffusion_step` effectuant un pas de diffusion non linéaire sur une image. Le script `exercice_4` vous permettra de visaliser l'évolution de l'image au cours du temps. Vous constaterez que les contours de l'image initiale sont bien mieux préservées, en comparaison de la diffusion linéaire. Le degré de lissage et de préservation des contours peut en outre être contrôlé en arrêtant la diffusion après un nombre fixé d'itérations, ainsi qu'en manipulant les paramètres δ_t et λ. Essayez pour terminer d'utiliser la diffusion non linéaire pour débruiter une image tout en préservant ses contours (voir l'aide de la fonction `imnoise` pour ajouter du bruit sur une image).