TD et TP 3 : Problème d'inpainting et Gradient Conjugué.

```
In [1]: %pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

1. Outils de manipulation d'images, position du problème d'inpainting.

Voici tout d'abord une fonction permettant de lire une image en niveau de gris sous la forme d'un tableau F. La fonction `shape` permet de donner les tailles dans chaque direction.

```
In [ ]: F=imread('mystere.png')
        pixelsv,pixelsh=shape(F)
        print(pixelsv,pixelsh)
```

On a donc dans notre cas un tableau de taille 512×512, voyons à quoi ressemblent les premières lignes et colonnes.

```
In [ ]: F[:20,:5]
```

On voit qu'une image en niveau de gris est codée par des valeurs dans $[0, 1]$, correspondant à la luminosité du pixel considéré (0 pour noir, 1 pour blanc).

Voici maintenant comment afficher une image sur une figure dont on peut varier au préalable la taille. La fonction `gray()` sert à ce que toutes les images affichées par la suite le soient en niveaux de gris. Le dernier point-virgule sert à ne pas afficher la sortie de `axis` (qui retourne sinon la valeur des bornes des axes).

```
In [ ]: gray()
        figure(figsize=(6,6))
        imshow(F)
        axis('off');
```

Le problème d'« inpainting » est le suivant : on se donne une image dont certains pixels n'ont pas été transmis, ce sont ceux pour lesquels la valeur vaut exactement 0, et on aimerait « interpoler » entre ces pixels de façon à remplir ces zones indéterminées.

On modélise l'image par une matrice $F = (f_{i,j}) \in M_{n_1,n_2}(\mathbb{R})$, où $f_{i,j} \in [0, 1]$ est la valeur de la luminosité au pixel de la ligne i et colonne j, et n_1 (resp. n_2) est le nombre de lignes (resp. de colonnes) de pixels. Et on va noter $M = (m_{i,j})$ la matrice du masque, c'est à dire $m_{i,j} \in \{0, 1\}$ avec $m_{i,j}$ qui vaut 1 si le pixel a été transmis et 0 si le pixel n'a pas été transmis (lorsque $f_{i,j} = 0$). Ce sont les données du problème.

Pour des raisons pratiques, on notera $\overline{M} = (\overline{m}_{i,j}) = (1 - m_{i,j})$ le contraire du masque, c'est-à-dire dont les entrées valent 1 si le pixel n'a pas été transmis et 0 sinon.

Le but est de retrouver une image complète, qui corresponde le plus possible à l'image originale. Pour cela on cherche donc une matrice $U = (u_{i,j})$ qui soit telle que $u_{i,j} = f_{i,j}$ si le pixel a été transmis et qui soit la plus « lisse » possible, elle va minimiser une sorte de norme L^2 d'une discrétisation du gradient. On considère donc le problème d'optimisation suivant :

$$\inf_{U \in C} \frac{1}{2} \sum_{i=1}^{n_1-1} \sum_{j=0}^{n_2-1} (u_{i,j} - u_{i-1,j})^2 + \frac{1}{2} \sum_{i=0}^{n_1-1} \sum_{j=1}^{n_2-1} (u_{i,j} - u_{i,j-1})^2,$$

où l'ensemble des contraintes est

$$C = \{U \in M_{n_1,n_2}(\mathbb{R}) | \forall (i, j) \text{ tels que } m_{i,j} = 1, \text{ on a } u_{i,j} = f_{i,j}\}.$$

On a fait commencer les différents indices à 0, pour être cohérent avec Python.

(a-TD) Montrer que le problème peut être vu comme un problème d'optimisation sans contraintes par rapport à la variable $X = U \circledast \overline{M}$ où $\circledast$ est la multiplication élément par élément (c'est-à-dire que

$x_{i,j} = u_{i,j}\overline{m}_{i,j}$), où l'on peut identifier les tels X comme des éléments d'un espace vectoriel $E \subset M_{n_1,n_2}(\mathbb{R})$ de dimension n. Comment calculer la dimension n à partir de la matrice $\overline{M}$?

(b-TP) Dans le cas qui nous intéresse, la matrice F est donnée par le tableau `F` obtenu par lecture de l'image. Calculer le tableau correspondant à la matrice $\overline{M}$ à partir de `F`, et calculer la dimension n du problème d'optimisation. Quel est la proportion de pixels transmis ?

(c-TD) Pour $U \in C$, exprimer $u_{i,j}$ en fonction de $x_{i,j}$ et $f_{i,j}$, et montrer que le problème d'optimisation sans contraintes sur la variable X est en fait un problème de minimisation quadratique d'une fonction du type $\frac{1}{2}\varphi_2(X,X) + \varphi_1(X)$, où φ_2 est une forme bilinéaire symétrique positive $E \to \mathbb{R}$ et φ_1 une forme linéaire $E \to \mathbb{R}$.

Exprimer $\varphi_2(X,Y)$ et $\varphi_1(X)$ lorsque $X, Y \in E$ en fonction des $x_{i,j}$, $y_{i,j}$ et $f_{i,j}$.

Montrer que si $M \neq 0$ (au moins un pixel a été transmis), la forme φ_2 est définie positive : $\varphi_2(X,X) > 0$ si $X \in E \setminus \{0\}$.

On note $\langle \Delta, \Delta \rangle$ le produit scalaire canonique sur les matrices, qui induit un produit scalaire sur E que l'on notera de la même manière : $\langle G, H \rangle = \mathrm{Tr}(G^T H) = \sum_{i=0}^{n_1-1} \sum_{j=0}^{n_2-1} g_{i,j} h_{i,j}$.

On sait alors qu'on peut avoir $\varphi_2(X,Y) = \langle X, A(Y) \rangle$ et $\varphi_1(X) = \langle B, X \rangle$ où $B \in E$ et A est un opérateur linéaire symétrique défini positif $E \to E$ (c'est à dire que $\langle A(X), Y \rangle = \langle X, A(Y) \rangle$, pour $X, Y \in E$, et que $\langle X, A(X) \rangle > 0$ pour $X \neq 0$), et qu'on a donc une solution unique au problème de minimisation. Quelle serait la taille de la matrice pour stocker A ?

(d-TD) Pourquoi cherche-t-on à pouvoir calculer $A(X)$, sans stocker X sous la forme d'un vecteur et A sous la forme d'une matrice ?

Montrer que $\varphi_2(X,Y) = \langle D_{n_1} X, D_{n_1} Y \rangle + \langle X D_{n_2}^T, Y D_{n_2}^T \rangle$ où D_{n_i} est une matrice nulle sauf sur la diagonale et la première diagonale inférieure. En déduire une expression de $A(X)$ faisant intervenir les matrices X, $D_{n_1}^T D_{n_1}$, $D_{n_2}^T D_{n_2}$ et la multiplication élément par élément $\circledast$ avec $\overline{M}$.

Obtenir de la même manière une expression de B faisant intervenir les matrices F, $D_{n_1}^T D_{n_1}$, $D_{n_2}^T D_{n_2}$ et la multiplication élément par élément $\circledast$ avec $\overline{M}$.

(e-TP) On cherche à obtenir une méthode efficace pour calculer $A(X)$.

- Calculer $D_k^T D_k$ pour des petites valeurs de k. Pour des tableaux à deux dimensions, la multiplication matricielle est donnée par la fonction `dot` : `dot(G,H)` renvoie la multiplication matricielle des tableaux `G` et `H`, à condition que `G` ait autant de lignes que `H` a de colonnes. La transposition est donnée par la fonction `transpose`.
- Y a-t-il un intérêt à stocker $D_{n_1}^T D_{n_1}$, et $D_{n_2}^T D_{n_2}$?
- Trouver une méthode pour calculer rapidement $D_k^T D_k X$ pour X une matrice de taille $k \times \ell$ en additionnant et soustrayant seulement des blocs de taille $k - 1 \times \ell$, et sans utiliser de multiplication matricielle. On pourra commencer par voir comment faire seulement $D_k X$, puis ensuite comment calculer rapidement $D_k^T Y$.
- Vérifier le résultat en prenant un X aléatoire pour des petites valeurs de k et ℓ.
- Évaluer le temps de calcul des deux méthodes pour des k et ℓ plus grands. On pourra utiliser la commande `%%timeit` de IPython qui évalue le temps mis à exécuter les commandes d'une cellule (exemple ci-dessous).

```
In [24]: %%timeit
         N=100
         G=randn(N,N)
         dot(transpose(G),G)

100 loops, best of 3: 7.28 ms per loop
```

(f-TP) Dans le cas du problème qui nous intéresse, programmer la fonction $X \mapsto A(X)$ de manière efficace par rapport à ce qui a été dit dans la question **(e)**. Évaluer également la matrice B de la même manière.

2. Résolution par méthode du gradient conjugué

(a-TP) Programmer la méthode de gradient conjugué en donnant A comme argument sous la forme d'une fonction. On pourra se donner un nombre maximal d'itérations, et prendre comme critère d'arrêt que

la norme du gradient $\|\nabla f(x_k)\|$ est inférieure à $\varepsilon\|\nabla f(x_0)\|$, où ε est une tolérance relative donnée. Ceci permet d'avoir un critère qui soit cohérent même quand la dimension change grandement.

(b-TP) Tester la méthode en prenant pour A une matrice symétrique définie positive de petite taille (il faudra donc aussi définir la fonction $X \mapsto A(X)$ pour pouvoir utiliser la méthode). On pourra par exemple prendre N une matrice aléatoire pas forcément symétrique (ni même carrée, mais dans ce cas il faut qu'il y ait plus de lignes que de colonnes), et prendre pour A la matrice $N^T N$. On peut calculer le nombre de conditionnement (la plus grande valeur propre divisée par la plus petite) avec la fonction `cond`.

Observer la convergence de la norme du gradient vers 0 (on pourra faire que la méthode renvoie la liste des $\|\nabla f(x_k)\|$ en plus du résultat final). Pourquoi la méthode ne renvoie-t-elle pas exactement 0 au bout de n itérations ?

(c-TP) Appliquer la méthode sur le modèle de la première partie et afficher l'image correspondant à la solution.

3. Résolution par méthode de descente de gradient à pas optimal et comparaison.

(a-TP) Programmer comme dans la partie précédente la méthode de descente de gradient à pas optimal dans le cas particulier de la minimisation d'une fonction quadratique $\frac{1}{2}\langle X, A(X)\rangle + \langle b, X\rangle$.

(b-TP) L'appliquer aux mêmes cas test que dans la partie précédente, et comparer les performances des méthodes.

4. Méthode de pénalisation

On reprend les notations de la première partie. Plutôt que de chercher à résoudre le problème d'optimisation sous contraintes que $U \in C$, on va essayer de résoudre le problème d'optimisation suivant

$$\inf_{U\in M_{n_1,n_2}(\mathbb{R})} \frac{1}{2}\sum_{i=1}^{n_1-1}\sum_{j=0}^{n_2-1}(u_{i,j}-u_{i-1,j})^2 + \frac{1}{2}\sum_{i=0}^{n_1-1}\sum_{j=1}^{n_2-1}(u_{i,j}-u_{i,j-1})^2 + \frac{1}{2\varepsilon}\sum_{i=0}^{n-1}\sum_{j=0}^{n_2-1} m_{i,j}(u_{i,j}-f_{i,j})^2.$$

(a-TD) À quoi sert le dernier terme, appelé terme de pénalisation, rajouté par rapport à la première partie ? Que se passe-t-il si le paramètre de pénalisation ε est très petit ?

(b-TD) Montrer avec les mêmes méthodes que dans la première partie, que ce problème revient à minimiser $\frac{1}{2}\langle U, A_\varepsilon(U)\rangle + \langle B_\varepsilon, U\rangle$ avec A un opérateur linéaire symétrique défini positif de $M_{n_1,n_2}(\mathbb{R})$ dans lui-même, et que l'on peut calculer $A_\varepsilon(U)$ de manière rapide sans utiliser de multiplication matricielle.

(c-TP) Utiliser la méthode de gradient conjugué pour trouver une approximation du U qui minimise la fonction précédente, et afficher l'image résultat, pour différentes valeurs de ε. Comparer « visuellement » les résultats des deux méthodes.

5. Minimisation différente pour l'inpainting

On cherche à éviter certains problèmes qui sont liés au fait que la minimisation est quadratique (en particulier, les bords sont mal rendus). On modifie la fonction à optimiser, et on s'intéresse au problème suivant :

$$\inf_{U\in C} J(U), \qquad \text{avec } J(U) = \frac{1}{2}\sum_{i=1}^{n_1-1}\sum_{j=1}^{n_2-1}\sqrt{\delta + (u_{i,j}-u_{i-1,j-1})^2 + (u_{i,j-1}-u_{i-1,j})^2},$$

où δ est un paramètre de régularisation qu'on aimerait prendre assez petit, où sa version par pénalisation

$$\inf_{U\in M_{n_1,n_2}(\mathbb{R})} J(u) + \frac{1}{2\varepsilon}\sum_{i=0}^{n-1}\sum_{j=0}^{n_2-1} m_{i,j}(u_{i,j}-f_{i,j})^2.$$

(a-TD) En utilisant les mêmes méthodes que dans la première partie, montrer que dans le premier cas on peut se ramener à un problème de minimisation sans contrainte par rapport à la variable $X \in E$.

Pourquoi ne peut-on plus appliquer la méthode de gradient conjugué pour ces deux problèmes ? Quelle méthode pourrait-on prendre ?

(b-TD) Calculer la dérivée partielle par rapport à $x_{i,j}$ (ou à $u_{i,j}$ dans le cas de la pénalisation) de la fonction à optimiser. En déduire qu'on peut exprimer le gradient sans faire de multiplication matricielle.

(c-TP) Programmer la méthode pour un des deux problèmes (ou les deux), et comparer « visuellement » le rendu des images correspondant au minimum de la fonction, par rapport aux résultats obtenus dans les parties précédente.

Comparer également le nombre d'itérations nécessaires dans les différents cas.