

# Une classe Image pour Matlab

D. Legland

19 avril 2011

Cette note décrit une classe « Image » qui a pour but de faciliter la manipulation des images sous Matlab. En particulier, les informations de type, de dimension spatiale sont plus facilement accessibles. Il est aussi possible de stocker des informations de calibration spatiale à l'image, ce qui facilite l'affichage des images acquises avec des résolutions différentes.

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Nature des images</b>	<b>3</b>
2.1	Dimension . . . . .	3
2.2	Types de pixels . . . . .	4
2.3	Calibration spatiale . . . . .	4
2.4	Information de visualisation . . . . .	4
2.5	Meta-données . . . . .	5
<b>3</b>	<b>Champs de classe</b>	<b>5</b>
3.1	Données brutes . . . . .	5
3.2	Informations sur l'image . . . . .	6
3.3	Calibration spatiale . . . . .	6
3.4	Orientation . . . . .	6
<b>4</b>	<b>Création des images</b>	<b>6</b>
4.1	Constructeur Image . . . . .	6
4.2	Création à partir d'un tableau . . . . .	7
4.3	Lecture depuis un fichier . . . . .	7
4.4	Création d'une image couleur . . . . .	7
<b>5</b>	<b>Manipulation des images</b>	<b>7</b>
5.1	Taille de l'image . . . . .	7
5.2	Accès et modification des valeurs de l'image . . . . .	8
5.3	Calibration spatiale . . . . .	9

5.4	Filtrage . . . . .	9
5.5	Affichage . . . . .	9
<b>6</b>	<b>Chose à rajouter ou en stand-by</b>	<b>10</b>
6.1	Gestion des dimensions . . . . .	10
6.2	Classe utilitaire Images . . . . .	11

# 1 Introduction

La plateforme Matlab est très performante pour traiter les images, mais manque parfois de souplesse pour manipuler les données. En particulier :

- les dimensions 1, 2 et 3 correspondent respectivement aux coordonnées y, x, et z, ce qui est peu intuitif et pénible pour les conversions
- pour les images couleur, la couleur est gérée comme une dimension, ce qui rend la manipulation des pixels laborieuse, surtout pour les images 3D (indexées en y, x, c, et z)
- pas de gestion de la résolution des images, ce qui vite pénible en 3D où la résolution est souvent différente pour l'axe z
- pas de gestion des meta-données
- l'indexation des pixels à partir de 1 n'est pas très heureuse, notamment pour les conversions de coordonnées
- les transformations géométriques d'images ne sont pas intuitives, et semblent difficile à étendre au 3D ou à d'autres types de transformations

Ce document présente une classe « Image » qui a pour but de simplifier la manipulation des images matricielles sous Matlab. Les objets qui instancient cette classe sont manipulés par référence, au contraire des tableaux ou structure de données, ce qui permet de limiter la consommation mémoire.

La section suivante commence par faire l'inventaire des images rencontrées couramment. La section suivante présente la structure de la classe, c'est à dire les champs de classes. Les sections suivantes présentent les fonctions permettant la création, et la manipulation des objets de type « Image ».

## 2 Nature des images

On se restreint pour le moment aux images matricielles. Chaque image est donc représentée par un tableau rectangulaire de plusieurs dimensions. Les dimensions du tableau pouvant représenter des dimensions spatiales, ou autres (longueur d'onde, temps...).

Les images autres que matricielles qui pourraient être traitées dans un deuxième temps :

- vectorielles (ensemble de primitives géométriques)
- piles d'images, avec rotations différentes pour chaque pile

### 2.1 Dimension

Les images les plus typiques sont des images 2D. On manipule régulièrement des images 3D. Le canal (couleur, ou spectral) peut être vu comme une dimension additionnelle. On peut avoir à manipuler des images en fonction du temps, ce qui rajoute une dimension. Idéalement il faudrait aussi inclure les images 1D afin d'être complet.

On se limite donc aux images de dimension 5, avec les dimensions correspondant aux coordonnées x, y, z, canal, et temps respectivement.

Notons qu'il est possible ainsi de représenter des coupes 2D tout en gardant le positionnement de la coupe dans l'espace 3D.

## 2.2 Types de pixels

Le type de pixel peut varier. On a d'une part des données scalaires :

- binaire
- niveaux de gris, codés sur 8, 12, 16... bits
- intensités, stockées sur des flottants en simple ou double précision
- labels, stockés en général en 16 bits
- autre : une carte d'orientation, une valeur de périmètre... -> peut être codé avec un type double.

On manipule aussi couramment des pixels représentés non plus par une, mais par un ensemble de valeurs :

- complexe, par exemple pour le résultat d'une transformation de Fourier
- couleur (RGB, HSV...)
- spectral

Pour certains cas, on peut même penser à des types composites. Par exemple, les modes HSV représentent 3 bandes qui n'ont pas les mêmes ordres de grandeur.

## 2.3 Calibration spatiale

On distingue deux types de référencement des coordonnées des pixels :

- soit les pixels sont stockées sur les sommets d'une grille : cela a l'avantage d'avoir des coordonnées entières pour les pixels, et donc de faciliter leur repérage (ex : si on dessine un point en coordonnées (3,5), il sera situé au centre du pixel correspondant)
- soit on considère que les pixels sont situés au centre d'un carré ou cube. L'avantage est que l'affichage est plus simple à gérer (un côté de carré zoom\*zoom). Re-échantillonner les images est peut-être aussi plus simple.

Pour la calibration spatiale, l'approche dans le logiciel ITK est d'intégrer les informations à la structure de données image. Idée à garder a priori.

Les deux infos à garder absolument :

- la taille du pixel dans chaque dimension
- la position du premier pixel

Ces deux informations sont en unité métrique, et correspondent à un tableau 1D avec autant d'éléments que le nombre de dimensions spatiales de l'image.

Pour l'orientation, des outils comme ITK passent par la matrice des cosinus. A voir pour une version ultérieure. On peut aussi imaginer un offset : l'origine ne correspond pas au pixel d'indice (0,0), mais à un autre (le pixel central, un autre coin...).

## 2.4 Information de visualisation

En gros, comment représenter les pixels de l'image. Il s'agit d'une application qui à une valeur de l'image (NDG, double, complexe...) associe une valeur d'affichage (en niveaux de gris ou en couleurs RGB). La transcription dépend du type d'image.

Pour une image en niveaux de gris, plusieurs paramètres peuvent être pris en compte :

- une valeur de gamma
- une fonctions linéaire (brillance + contraste, cf. ImageJ)

- la palette de couleur utilisée, si l'image est indexée

Pour une image d'intensité affichée en niveaux de gris, on doit en général spécifier la dynamique d'affichage (min et max), ce qui permet de recalibrer l'affichage pour des valeurs entre 0 et 255.

Pour une image multivariée (spectre, complexe, couleur...), on peut passer par une fonction qui détermine la couleur d'affichage en fonction d'un vecteur. Par exemple, on peut afficher le module d'un nombre complexe.

## 2.5 Meta-données

Quelques infos qu'il est parfois bon d'avoir sous la main :

- nom du fichier
- nom de l'image
- min et max des valeurs des pixels (pour images scalaires)
- résolution, pour avoir une correspondance entre coord pixels et coord physiques.
- nom des axes
- unité de la valeur des pixels

Matlab utilise une fonction spéciale pour lire les infos d'un fichier et stocke le résultat dans une structure.

## 3 Champs de classe

On présente ici les différents champs de la classe Image, en les regroupant par thème. Ces différents champs sont supposés être initialisés lors de la construction de l'objet image, et ne devraient pas être modifiable a priori.

### 3.1 Données brutes

**data** Contient le tableau de données de l'image. Pour éviter les changements d'index de matlab, l'idée est de stocker l'image telle que les dimensions correspondent dans l'ordre à x, y, et éventuellement z. Pour des images multivariées (tensorielles), on rajoute une dimension, à la fin. Pour des images temporelles, on rajoute encore une dimension. On traite donc un tableau de dimension 5, ce qui permet de gérer la majorité des cas possibles.

**dataSize** La dimension totale de l'image. Il est nécessaire de le stocker pour éviter les problèmes de dimensionnement. Il s'agit d'un vecteur ligne de 5 colonnes.

**elementSize** la taille du tableau représentant chaque pixel ou voxel. Vaut 1 dans le cas d'une image scalaire, N dans le cas d'une image vectorielle, avec N étant le nombre de composantes du vecteur. On peut aussi imaginer des images où chaque pixel est une matrice (ex : matrice d'orientations), mais l'occupation mémoire peut augmenter rapidement.

### 3.2 Informations sur l'image

**type** le type d'image, c'est à dire la manière d'interpréter le ou les canaux de l'image. Peut prendre une des valeurs suivantes : 'binary', 'grayscale', 'intensity', 'label', 'color', 'vector', 'complex', 'unknown'.

**channelNames** le nom de chaque canal (utile surtout pour les images multi-canaux). Peut être vide, ou un tableau de chaînes. Ex : {'intensity'}, {'distance'}, {'red', 'green', 'blue'}, {'real', 'imaginary'}...

### 3.3 Calibration spatiale

On a ensuite des informations de calibration spatiale, un ensemble de données permettant le passage entre les coordonnées physique et l'index

**origin** les coordonnées physique du premier pixel de l'image. Stocké dans un vecteur ligne de  $d$  éléments.

**spacing** l'espacement, en unités physique, entre deux éléments successifs. Stocké dans un vecteur ligne de  $d$  éléments.

**unitName** le nom de l'unité physique, stocké dans une chaîne de caractères. Exemples : « microns », « millimètres », « nm »...)

**calibrated** indicateur booléen qui est mis à VRAI si la calibration doit être prise en compte.

### 3.4 Orientation

Toujours concernant la calibration spatiale, on stocke des informations sur la manière de s'orienter dans l'image.

**axisNames** le nom de chacun des axes.

**(?)anatomicalOrientation** un code de trois lettres décrivant l'orientation anatomique de l'objet représenté dans l'image.

## 4 Création des images

La bibliothèque permet de créer des images à partir d'un tableau contenant les données, de charger l'image contenue dans un fichier, ou de créer des images en combinant des images déjà en mémoire.

### 4.1 Constructeur Image

Pour le moment, ce constructeur est protégé. La création d'une image se fait donc via une des différentes méthodes statiques. Ceci permet normalement de faciliter les modifications en cas de changement des implémentations.

## 4.2 Création à partir d'un tableau

La classe Image définit une méthode statique « create », qui crée un objet Image à partir d'un tableau Matlab. Cette fonction déduit la taille, la dimension, et le type de l'image en fonction de la taille du tableau de données.

## 4.3 Lecture depuis un fichier

Une fonction statique « read ».

Question : comment gérer l'ajout de formats de fichiers ?

## 4.4 Création d'une image couleur

Une fonction createRGB, à laquelle on passe de 1 à 3 arguments, et qui renvoie l'image RGB correspondante.

# 5 Manipulation des images

On a besoin d'opérations de base pour accéder aux valeurs des pixels, éventuellement les modifier, et connaître les informations générales de l'image (taille, nature des éléments...). On aimerait aussi avoir des fonctions qui se comportent de la même manière quelle que soit la dimension (moyenne, min, max... de l'image).

## 5.1 Taille de l'image

Fonctions de base pour connaître l'image.

**size**

Renvoie la taille de l'image, dans un vecteur ligne dont la longueur est égale à la dimension de l'image.

**dim**

Renvoie le nombre de dimensions -> reprendre ndims qui est casse-pieds pour les tableaux 1D. partir sur outerDimension ?

**innerDimension (innerDim)**

Renvoie le nombre de coordonnées pour lesquelles on a plus de 1 coupes.

**outerDimension (outerDim)**

Renvoie la dimension dans laquelle est incluse l'image. Est supérieur ou égal à la innerDimension.

## 5.2 Accès et modification des valeurs de l'image

### 5.2.1 Utilisation de l'indexage direct

Les fonctions `subsref` et `sunsasgn` ont été surchargées, ce qui permet un accès direct (en lecture et en écriture) aux valeurs de l'image.

Permet :

- renvoyer la valeur d'un pixel à une position donnée
- crop image en spécifiant une plage.

### 5.2.2 Utilisation de fonctions d'accès

#### **pixel**

Renvoie un pixel, ou un ensemble de pixels, sous la forme d'un scalaire, ou d'un tableau à plusieurs dimensions.

#### **subImage**

Extrait une sous-image, soit de même dimension, soit de dimension réduite. Le résultat est encore un objet image.

### 5.2.3 Statistiques de base

Ces fonctions renvoient des valeurs calculées sur l'ensemble des pixels de l'image. Elle s'appliquent à des images scalaires, ou vectorielles.

**mean** renvoie la valeur moyenne

**min** renvoie la valeur minimum

**max** renvoie la valeur maximum

**median** renvoie la valeur médiane

**var** renvoie la variance des valeurs

**std** renvoie l'écart-type des valeurs de l'image

### 5.2.4 Opération arithmétiques sur l'ensemble des pixels

**plus** ajoute une constante à chaque pixel

**minus** enlève une constante à chaque pixel

**times** multiplie chaque pixel par une constante

**divide** divise chaque pixel par une constante

### 5.2.5 Opérations entre deux images

**plus** additionne les valeurs des pixels de deux images, qui doivent avoir des pixels de même type

**minus** soustrait les valeurs des pixels de deux images, qui doivent avoir des pixels de même type



### 5.3 Calibration spatiale

Il nous faut quelques méthodes de conversion entre les deux systèmes de coordonnées utilisés :

- le système des indices, entre 1 et le nombre d'éléments dans la direction correspondante (on garde la convention Matlab)
- le système physique, en unité utilisateur, qu'on appelle « positions ».

On peut imaginer les fonctions suivantes :

**getSpacing/setSpacing** pour changer l'espacement entre les pixels ou voxels

**getOrigin/setOrigin** pour changer la position spatiale du premier élément du tableau

**getUnitName** pour connaître le nom de l'unité utilisée

**pointToIndex** conversion entre les coordonnées physiques et les coordonnées index

**indexToPoint** conversion entre les coordonnées index et les coordonnées physiques

### 5.4 Filtrage

#### 5.4.1 Transformations géométriques basiques

Modification de la géométrie par rotation, symétries...

**flip**

Renverse une des dimensions

**rotate90**

Rotation de 90 degrés autour d'un des axes

### 5.5 Affichage

On veut pouvoir afficher une image avec différents niveaux de zoom. Choix entre 2 possibilités.

#### 5.5.1 Pixels entre les bornes

Pour chaque pixel, on affiche un rectangle de la couleur du pixel entre les coordonnées  $(i, j)$  et  $(i + 1, j + 1)$ . Méthode classique de représentation des images sur écran matriciel.

L'avantage est que le coin supérieur de l'image est situé aux coordonnées  $(0, 0)$ , et que l'image occupe à l'écran un rectangle dont la position se calcule facilement :  $[0; M_i] \cdot zoom$ , où  $M_i$  est le nombre de pixels dans la dimension  $i$ .

Pour convertir les coordonnées écran en coordonnées pixel, il suffit de tronquer, après avoir divisé par le facteur de zoom.

### 5.5.2 Pixel centré sur les bornes

L'idée est de positionner le centre de chaque pixel aux coordonnées physiques du pixel. C'est ce qui est utilisé dans ITK.

L'avantage est que l'on a une meilleure correspondance entre les coordonnées physiques et le centre des pixels, ce qui est préférable si on veut mixer différentes images ou alors des images avec des objets vectoriels (résultats de segmentation par exemple).

Un pixel sera affiché dans un rectangle de coordonnées :

$$\prod_{i=0,\dots,d} \left[ x_i + o_i - \frac{w_i}{2}; x_i + o_i + \frac{w_i}{2} \right]$$

où les  $x_i$  sont les coordonnées du pixel,  $o_i$  l'origine de la grille, et  $w_i$  la taille de la grille selon chaque dimension.

Pour convertir les coordonnées écran en coordonnées pixel, l'opération semble un peu plus compliquée que dans le cas d'un pixel dans les mailles de la grille, mais reste abordable. De plus, sous Matlab on peut déléguer à la gestion des axes.

### 5.5.3 Conclusion

Il vaut mieux choisir des pixels centrés sur la grille.

## 6 Chose à rajouter ou en stand-by

On peut stocker les infos nécessaire pour l'interprétation et l'affichage des valeurs

**lut/dynamic** Pour l'affichage. Stocké sous forme de tableau  $2^G \times 3$ .

**minValue**, **maxValue** peuvent être utiles pour certains calculs. Nécessitent d'être actualisés si on change les valeurs de certains pixels. à réfléchir.

**getInfos** renvoie les meta-données associées à l'image.

### 6.1 Gestion des dimensions

Comment gérer les coupes 2D dans un espace 3D ? C'est techniquement faisable, mais il faut préciser la notion de dimension.

Par exemple, une coupe XZ aura comme taille  $[N_x \ 1 \ N_z]$ . Si on applique une fonction « squeeze » ou équivalent, la taille deviendra  $[N_x \ N_z \ 1]$ , et la dimension sera égale à 2.

Dans ce cas, comment différencier une image 2D et une coupe XY ? Dans les deux cas, la taille est  $[N_x \ N_y \ 1]$ .

Une solution envisageable est de stocker le nombre de dimensions dans une variable « nd », qui contient le nombre de dimensions spatiales. Ainsi, si nd vaut par exemple 3, alors la dimension de l'image sera de 3 même si elle se limite à un point, qui correspond à un simple voxel.

## 6.2 Classe utilitaire Images

Une classe pour fournir des méthodes statiques telles que le chargement d'une image, ou quelques filtres standards.

**read** pour charger une image depuis un fichier

**write** pour sauver les données image dans un ou plusieurs fichiers.