Instructions pour T.P. déconvolution d'image

Éric Thiébaut, Mireille Louys & Jonathan Léger

version du 28 janvier 2016

1 Objectifs du T.P.

1.1 Environnements de travail

Pour les besoins du T.P. vous avez besoin d'un langage de traitement des données (NumPy, Yorick, IDL/GDL/PV-Wave, Matlab/Octave/Scilab, Julia, *etc.*).

Sur la clef USB (répertoire `TP-invpb`) vous avez :

- `data` – Des données à traiter.
- `python` – Scripts pour NumPy.
- `yorick` – Scripts pour Yorick ;
- `matlab` – Scripts pour Matlab/Octave ;
- `soft/Anaconda*` — Différentes versions d'Anaconda (`https://www.continuum.io/downloads`) qui fournit NumPy pour différents types de machines.

1.2 Lire et afficher les données

Pour la déconvolution d'image, vous avez besoin d'une image de l'objet (*e.g.*, `saturn.fits`) et d'une image d'une source de référence (*e.g.*, `saturn_psf.fits`).

Utilisez l'environnement de travail pour lire ces images et les afficher. Référez-vous à la section spécifique au langage de traitement que vous allez utiliser pour effectuer ces premières étapes.

1.3 Filtre de Wiener approché

1. Lire l'image de Saturne par le HST (`data/saturn.fits` ou `data/saturn.mda`) et la réponse impulsionnelle (`data/saturn_psf.fits` ou `data/saturn_psf.mda`).
2. Afficher ces deux images.
3. Dans un premier temps, nous allons utiliser une version simplifiée du filtre de Wiener :

$$\widetilde{x}_k^{[\mu]} = \frac{\widetilde{h}_k^\star \, \widetilde{y}_k}{|\widetilde{h}_k|^2 + \mu} \tag{1}$$

Expliquer ce que représentent les différentes variables de la formule ci-dessus. Quel est le rôle du terme μ ? Donner le pseudo-code correspondant à ce filtrage.

4. Discuter des modalités d'application sur l'exemple de l'image de Saturne. Coder une fonction effectuant ce filtrage et régler visuellement la valeur de μ de façon optimale.

1.4 Approche inverse par méthode itérative

Nous allons résoudre le problème de la déconvolution en minimisant la fonction de coût :

$$\phi(\boldsymbol{x}) = \phi_{\mathsf{data}}(\boldsymbol{x}) + \mu\,\phi_{\mathsf{prior}}(\boldsymbol{x}) \tag{2}$$

avec

$$\begin{aligned}\phi_{\mathsf{data}}(\boldsymbol{x}) &= (\mathbf{H}\cdot\boldsymbol{x} - \boldsymbol{y})^{\mathrm{t}}\cdot\mathbf{W}\cdot(\mathbf{H}\cdot\boldsymbol{x} - \boldsymbol{y}) &(3)\\ &= \sum_i w_i\left[(\boldsymbol{h}\star\boldsymbol{x})_i - y_i\right]^2 &(4)\end{aligned}$$

où $\mathbf{H}$ est l'opérateur de convolution par $\boldsymbol{h}$ et

$$\begin{aligned}\phi_{\mathsf{prior}}(\boldsymbol{x}) &= \|\mathbf{D}\cdot\boldsymbol{x}\|^2 &(5)\\ &= \sum_{i,j}\left[x_{i+1,j} - x_{i,j}\right]^2 + \sum_{i,j}\left[x_{i,j+1} - x_{i,j}\right]^2 &(6)\end{aligned}$$

avec $\mathbf{D}$ un opérateur linéaire (de type différences finies).

1. En utilisant les expressions algébriques, montrer que le minimum de la fonction de coût est la solution d'un système linéaire d'équations de la forme :

$$\mathbf{A}\cdot\boldsymbol{x} = \boldsymbol{b} \tag{7}$$

   Donner les expressions de $\mathbf{A}$ et de $\boldsymbol{b}$.

2. Écrire un code NumPy/Yorick/Matlab/Julia qui calcule $\mathbf{A}\cdot\boldsymbol{x}$ étant donné $\boldsymbol{x}$. Il faut utiliser la FFT pour la convolution et implanter les opérateurs $\mathbf{H}$ et $\mathbf{H}^{\mathrm{t}}$. Pour appliquer l'opérateur $\mathbf{D}^{\mathrm{t}}\cdot\mathbf{D}$, la fonction $\boldsymbol{x}\mapsto\mathbf{D}^{\mathrm{t}}\cdot\mathbf{D}\cdot\boldsymbol{x}$ est fournie pour les différents langages (par exemple, la fonction `deconv_dtd` dans le fichier `deconv_dtd.m` pour Matlab).

3. La méthode des gradients conjugués linéaires est un algorithme simple et efficace (rapide et peu gourmand en mémoire) pour résoudre de façon itérative un système linéaire d'équations de la forme :

$$\mathbf{A}\cdot\boldsymbol{x} = \boldsymbol{b} \tag{8}$$

   où le « vecteur » $\boldsymbol{x}$ contient les inconnues et la matrice $\mathbf{A}$ est symétrique définie positive. L'algorithme des gradients conjugués linéaires est donné par la Figure 1. Les ingrédients de la méthode sont :

   - une fonction réalisant l'opération $\boldsymbol{x}\mapsto\mathbf{A}\cdot\boldsymbol{x}$;
   - une estimation initiale de $\boldsymbol{x}$ (une image remplie de zéros ou le résultat d'une recosntruction précédente feront l'affaire) ;
   - le « vecteur » $\boldsymbol{b}$;

   La fonction `conjgrad` en NumPy (ou `cg` dans `tp-utils.i` en Yorick ou `conjgrad` en Matlab dans `conjgrad.m`) permet d'appliquer les gradients conjugués.

Gradients conjugués

initialisation:
  compute $\boldsymbol{r}_0 = \boldsymbol{b} - \mathbf{A} \cdot \boldsymbol{x}_0$ for some initial guess $\boldsymbol{x}_0$
  let $k = 0$
until convergence do
  $\rho_k = \boldsymbol{r}_k^{\mathrm{t}} \cdot \boldsymbol{r}_k$
  if $k = 0$, then
    $\boldsymbol{p}_k = \boldsymbol{r}_k$
  else
    $\beta_k = \rho_k / \rho_{k-1}$
    $\boldsymbol{p}_k = \boldsymbol{r}_k + \beta_k \boldsymbol{p}_{k-1}$
  endif
  $\boldsymbol{q}_k = \mathbf{A} \cdot \boldsymbol{p}_k$
  $\alpha_k = \rho_k / (\boldsymbol{p}_k^{\mathrm{t}} \cdot \boldsymbol{q}_k)$ (optimal step size)
  $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$
  $\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k \boldsymbol{q}_k$
  $k \leftarrow k + 1$
done

Figure 1: Algorithme des gradients conjugués linéaires.

2 Utilisation de Matlab/Octave

Les instructions du T.P. pour les fans de ce langage sont dans le fichier `tp-matlab.pdf`.

3 Utilisation de NumPy

3.1 Installation de NumPy

Pour faire le T.P. en NumPy (*Numerical Python*) il vous faut installer un interpréteur Python et un certain nombre d'extensions ou de modules. Deux possibilités :

1. Vous pouvez installer une distribution comme Anaconda (https://www.continuum.io/downloads) ou Canopy (https://store.enthought.com/downloads/) qui fournissent Python et une bibliothèque très riche d'extensions dont NumPy. Inutile de télécharger Anaconda, si le répertoire `soft` contient déjà une version qui convient à votre type d'ordinateur. Sous Linux, l'installation d'Anaconda se fait par :

   ```
   bash Anaconda-2.4.0-Linux-x86_64.sh
   ```

   Pour Canopy, il faudra utiliser le gestionnaire de paquets pour installer astropy.

2. Si vous utilisez une autre distribution de Python que celle établie par Anaconda, vous aurez besoin de NumPy, de MatplotLib et du module AstroPy (http://www.astropy.org/), par exemple sous Ubuntu :

   ```
   sudo apt-get install python-numpy python-matplotlib python-astropy
   ```

   IPython (http://ipython.org/) n'est pas indispensable mais c'est un environnement interactif pour utiliser Python qui vous simplifiera la vie. Il est inclus dans Anaconda, pour l'installer sous Ubuntu :

   ```
   sudo apt-get install ipython
   ```

3.2 Démarrage de NumPy pour le T.P.

Il faut lancer l'interpréteur Python ou IPython, le plus simple est de le faire depuis le répertoire où se trouve le fichier `python/tp_utils.py` :

```
cd python
ipython
```

Un certain nombre de fonctions utiles pour le T.P. sont disponibles dans le module `tp_utils`. La toute première chose à faire est d'importer ces fonctions :

```
from tp_utils import *
```

3.3 Lire et afficher les données avec NumPy

Le court exemple ci-dessous indique comment lire les fichiers qui contiennent les images de l'objet et d'une source de référence et comment les afficher.

```
# Chargement des routines de base
from tp_utils import *

# Lecture de l'image
y = fits_read("../data/saturn.fits")
```

```
h = fits_read("../data/saturn_psf.fits")

# Affichage
figure()
imshow(y, vmin=0)     # affichage avec coupure a zero

# Affichage avec une table de couleur differente
clf()           # on efface la figure
imshow(y, vmin=0, cmap="gist_stern")

# Affichage de la PSF dans une autre fenetre
figure()            # nouvelle fenetre
imshow(h, vmin=0, cmap=...) # où ... est le nom d'une table de couleurs
```

Vous êtes prêt pour attaquer la suite... À toutes fins utiles, vous trouverez ci-après un bref récapitulatif de la syntaxe de base de Python et des fonctions les plus utiles (le fichier python/tp_utils.py définit pour vous un certain nombre de *raccourcis* pour vous faciliter la vie).

3.4 Précis de NumPy

```
from tp_utils import *  # initialization
x = expr                  # donner une valeur a une variable

# Pour les variables stockant des tableaux multi-dimensionnels :
x.size          # nombre d'elements dans x
x.shape         # dimensions de x
x[i]            # le i-eme element (a partir de 0) de x
x[i,j]          # un element d'un tableau 2D
x[:,j]          # la j-eme colonne
x[i1:i2,j1:j2] # une sous-image (i1≤i<i2 et j1≤j<j2)
x[test]         # le sous-tableau pour lequel 'test' est vrai
x[x <= 0]       # les elements de 'x' negatifs ou nuls

# Creation d'un tableau :
a = np.zeros((dim1, dim2, ...), dtype=dtype)
a = np.ones(shape, dtype=dtype)
a = np.empty(shape, dtype=dtype)

# Fonctions utiles pour le T.P. :
zeropad(x, shape)     # remplissage de zeros a une taille donnee
z = fft(x)            # FFT appliquee a x
x = ifft(z)           # FFT inverse
fftshift(x)           # recentrage FFT
ifftshift(x)          # recentrage inverse FFT
real(z)               # partie reelle de z
imag(z)               # partie imaginaire de z
conj(z)               # complexe conjugue de z
abs2(z)               # module carre de z
conjgrad(A, b, ...)   # gradients conjugues
DtD(x)                # calcule D^t . D . x
```

```
data = fits_read("filename")    # lire une image FITS
fits_write("filename", data)    # lire une image FITS

def f(arg1, arg2, ...):   # definir une fonction f
    ...                   # le corps de la fonction est indente
    return expr           # resultat

%run file.py          # execute le contenu du fichier file.py
execfile("file.py")   # idem sans IPython
```

4 Instructions pour Yorick

Consulter http://yorick.sourceforge.net pour plus d'informations sur Yorick.

Pour l'essentiel, la syntaxe de Yorick (les expressions, les structures de contrôle, les boucles, les tests, etc.) sont semblables au C. Les principales différences sont :

- les variables sont dynamiquement typées et allouées ;
- le résultat d'une opération mettant en jeu des tableaux est un tableau (l'opération étant appliquée élément par élément) ; par exemple :
  - Yorick : `a = b + c*d;`
  - C : `for (i = 0; i < n; ++i) a[i] = b[i] + c[i]*d[i];`
- les indices des tableaux commencent à 1 et la liste des indices est donnée entre parenthèses ;
  - Yorick : `a(i,j,k)`
  - C : `a[k-1][j-1][i-1]`

4.1 Précis de Yorick

4.1.1 Général

```
#include "fichier.i"              // interprete le contenu de fichier.i
help, fn;                         // aide sur la fonction fn
info, expr;                       // information sur l'expression expr
about, "regex";                   // recherche d'aide
dbexit;                           // sortir du mode debug
x = expr;                         // donner une valeur a une variable
expr;                             // affiche le resultat de l'expression expr
numberof(x)                       // nombre d'elements dans x
dimsof(x)                         // dimensions de x
a = array(type, dim1, dim2, ...);// creation d'un tableau
i = where(test);                  // liste des indices ou test est vrai
indgen(n);                        // listes des indices de 1 a n
func f(arg1, arg2, ...) {         // creation d'une fonction f
    ...;
    return expr;
}
```

4.1.2 Syntaxe sur les tableaux

```
x(i)              // le i-eme element de x (i = 1 pour le premier)
x(i:j)            // les elements de i a j de x
x(i:j:s)          // les elements de i a j par pas s de x
x(0)              // le dernier element de x
x(-1)             // l'avant dernier element de x
x(*)              // tous les elements de x en un tableau 1-D
img(i,)           // la i-eme colonne de img (un tableau 2D)
img(,j)           // la j-eme ligne de img (un tableau 2D)
img(,avg:11:59:3) // la moyenne (avg) des lignes 11 a 59 par pas de 3
```

```
[]                  // rien ou vide
[1.0, 5.0, 15.0]    // un tableau 1-D de 3 valeurs
[[1,2,3],[4,5,6]] // un tableau 2-D de dimensions 3 par 2
x(dif,..)           // difference finie le long de la 1ere dimension
x(,dif,..)          // difference finie le long de la 2eme dimension
```

4.1.3 Graphismes

Note : la plupart des commandes graphiques s'appliquent à la fenêtre courante.

```
window, n;        // selectionner la fenetre n (la creer si
                  // elle n'existe pas)
show, img;        // afficher une image (simple)
show, img, n;     // afficher une image img dans la fenetre n
pli, img;         // afficher l'image img
plg, y;           // afficher la courbe
plg, y, x;        // afficher la courbe y(x)
fma;              // effacer la liste graphique
limits;           // re-initialiser les limites
limits, xmin, xmax, ymin, ymax; // fixer les limites
limits, square=0/1;       // axes x/y de meme aspect ratio
logxy, 0/1, 0/1;          // axes lineaires/logarithmiques
palette, "stern.gp";      // changer de palette de couleur
```

4.1.4 Utilitaires

```
img = fits_read("input.fits");  // lire un fichier FITS
fits_write, "output.fits", img; // ecrire un fichier FITS
zeropad(a, dims);               // etendre un tableau a par des zeros
abs2(z);                        // module carre de z
z = fft(x);                     // FFT de x
x = ifft(z);                    // FFT inverse de z (Hermitique)
fft_plg, y;                     // afficher une courbe en geometrie FFT
fft_pli, z;                     // afficher une image en geometrie FFT
u = fft_dist(dimsof(z));        // longueur des frequences spatiales
conj(z);                        // complexe conjugue de z
help, cg;                       // aide pour les gradients conjugues
histo_plot, a, binsize=0.1, color="red";
                                // afficher un histogramme des valeurs de a
```