Introduction à UNLocBoX et son utilisation en optimisation convexe

22 octobre 2025

1 Introduction générale

L'**UNLocBoX** (*UNiversal LOCal BOx*) est une boîte à outils libre développée à l'EPFL (École Polytechnique Fédérale de Lausanne) pour la **résolution de problèmes d'optimisation convexe non lisse**. Elle est conçue pour fonctionner sous MATLAB et GNU Octave, et permet d'expérimenter aisément avec des méthodes de type *proximal splitting* telles que :

— *Forward-Backward Splitting (FBS)*,
— *Douglas-Rachford*,
— *ADMM* (Alternating Direction Method of Multipliers),
— *SDMM* (Simultaneous Direction Method of Multipliers).

Ces algorithmes permettent de résoudre des problèmes du type :

$$\min_x \ f(x) + g(x)$$

où f est différentiable à gradient lipschitzien, et g est convexe mais éventuellement non différentiable. L'un des grands intérêts d'UNLocBoX réside dans son implémentation de nombreux **prox opérateurs** standards, tels que :

$$\text{prox}_{\lambda\|x\|_1}, \quad \text{prox}_{\lambda\|x\|_2^2}, \quad \text{prox}_{\lambda \text{TV}(x)}$$

pour la régularisation en norme L_1, L_2, et la variation totale (*Total Variation*).

2 UNLocBoX et optimisation convexe

En optimisation convexe, de nombreux problèmes pratiques peuvent être formulés sous la forme :

$$\min_{x \in \mathbb{R}^n} F(x) = \sum_{i=1}^{m} f_i(x)$$

où chaque f_i est convexe. UNLocBoX offre une interface unifiée pour résoudre ces problèmes en combinant les prox opérateurs de chaque fonction.

Les domaines d'application sont variés :

— débruitage et restauration d'images,
— séparation de signaux,
— reconstruction compressée,
— apprentissage parcimonieux.

3 Installation de Unlockbox Sous GNU Octave (Windows)

Unlockbox est un outil utilisable aussi bien sur Matlab que sur GNU Octave. Le procédé d'installation de cet outil sous GNU octave est la suivant :

- Télécharger le fichier *.zip* de unlockbox depuis le lien `https://github.com/epfl-lts2/unlocbox`
- Décompresser le fichier *.zip* issue du téléchargement
- Choisir un répertoire d'environnement où placer le dossier dézippé.
- Sauvegarder le chemin d'accès du contenudu dossier dézippé dans le presse-papier (ctrl + c)

Lorsque cette étape est réalisé, il faut maintenant pouvoir utiliser unlockbox, mais pour cela il faudra

- Ouvrir Octave
- Ouvrir la fenêtre de commande de Octave et taper les commandes :

Listing 1 – Appel de Unlockbox

```
>> %Ajouter le chemin vers le r pertoire de unlockbox
>> addpath(genpath('chemin_vers_unlockbox'));
>> %sauvegarder le chemin
>> savepath;
```

Ensuite pour l'utiliser depuis un code source, utiliser

```
1 init_unlockbox;
```

Listing 2 – initialisation de Unlockbox

4 Exemple d'utilisation : débruitage d'une image

L'exemple suivant illustre l'utilisation d'UNLocBoX pour le **débruitage d'image par optimisation convexe**. L'image est bruitée, puis régularisée par trois fonctions convexes :

- régularisation L_2 (Tikhonov),
- régularisation L_1,
- régularisation en variation totale (TV).

Notons que l'image ici est ***Exemple.png*** qui est la suivante :

Chargement de l'image, conversion en niveau de gris, bruitage et affichage

```
init_unlockbox;

pkg load image; %necc saire
% --- param tres utilisateur
img_file = 'Exemple.png';
sigma = 0.05;                    % amplitude du bruit gaussien (std)
lambda_tv = 0.08;                % poids TV
lambda_l1 = 0.06;                % poids L1
lambda_l2 = 0.06;                % poids L2 (Tikhonov via prox_l2)
maxit_solver = 200;

% --- lire image
I0 = im2double(imread(img_file));
if size(I0,3) == 3
  I0 = rgb2gray(I0);
end
[H, W] = size(I0);

% --- ajouter bruit gaussien
rng(0);  % pour reproductibilit
Inoisy = I0 + sigma * randn(size(I0));
Inoisy = min(max(Inoisy, 0), 1);

figure('Name','Original / Noisy','NumberTitle','off');
subplot(1,2,1); imshow(I0); title('Original');
subplot(1,2,2); imshow(Inoisy); title(sprintf('Noisy (\\sigma=%.3f)
   ', sigma));

% --- fonctions communes (fidelity = 0.5*||x-y||^2)
y = Inoisy;                      % mesure
f2.eval = @(x) 0.5 * norm(x(:) - y(:))^2;
f2.grad = @(x) (x - y);     % gradient of 0.5||x-y||^2 is (x-y)
```

Listing 3 – Travail 1 sur l'image

On obtient un 1 er résultat sous cette forme...

FIGURE 1 – Image en niveau de gris et son bruit gaussien

Puis Utilisation des méthodes de **TV régularisation**, **L1 régularisation et L2 régularisation** tous de *Unlockbox*, pour la minisation des erreurs de l'image bruitée pour aller sur une similitude de l'initiale.

```
% solver params
param_solver.maxit = maxit_solver;
param_solver.verbose = 1;    % 0 pour muet
param_solver.gamma = 1.0;    % step size (OK ici car Lipschitz
    constant L=1 for grad)

% --------------------------
% METHOD A: TV regularization
% minimize 0.5||x-y||^2 + lambda_tv * TV(x)
% --------------------------
param_tv.verbose = 0;
param_tv.maxit = 50;

f_tv.prox = @(x, tau) prox_tv(x, lambda_tv * tau, param_tv);  %
    prox for lambda*TV with step tau
f_tv.eval = @(x) lambda_tv * norm_tv(x);

x0 = y;  % init
[x_tv, info_tv] = forward_backward(x0, f_tv, f2, param_solver);

% --------------------------
% METHOD B: L1 regularization (sparsity on pixels)
% minimize 0.5||x-y||^2 + lambda_l1 * ||x||_1
% --------------------------
```

```
% prox_l1 applies soft-thresholding element-wise
f_l1.prox = @(x, tau) prox_l1(x, lambda_l1 * tau);
f_l1.eval = @(x) lambda_l1 * norm(x(:), 1);

[x_l1, info_l1] = forward_backward(x0, f_l1, f2, param_solver);

% --------------------------
% METHOD C: L2 regularization (Tikhonov)
% minimize 0.5||x-y||^2 + lambda_l2 * ||x||_2^2
% prox_l2 solves prox for weighted L2; with default A=Id and y=0
   gives shrinkage
% --------------------------
% prox_l2 signature: sol = prox_l2(x, gamma, param)
param_l2 = struct(); % default ok (A = Id, y = 0)
f_l2.prox = @(x, tau) prox_l2(x, lambda_l2 * tau, param_l2);
f_l2.eval = @(x) lambda_l2 * (norm(x(:),2)^2);

[x_l2, info_l2] = forward_backward(x0, f_l2, f2, param_solver);
```

Listing 4 – Utilisation des méthodes de Unlockbox pour la minisation

```
    % --------------------------
% Eval metrics: MSE / PSNR
% --------------------------
mse = @(A,B) mean((A(:)-B(:)).^2);
psnr_from_mse = @(m) 10*log10(1./m);

MSE_noisy = mse(I0, Inoisy);
MSE_tv    = mse(I0, x_tv);
MSE_l1    = mse(I0, x_l1);
MSE_l2    = mse(I0, x_l2);

fprintf('\n--- MSE / PSNR ---\n');
fprintf('Noisy: MSE=%.6f, PSNR=%.2f dB\n', MSE_noisy, psnr_from_mse
   (MSE_noisy));
fprintf('TV   : MSE=%.6f, PSNR=%.2f dB\n', MSE_tv,    psnr_from_mse
   (MSE_tv));
fprintf('L1   : MSE=%.6f, PSNR=%.2f dB\n', MSE_l1,    psnr_from_mse
   (MSE_l1));
fprintf('L2   : MSE=%.6f, PSNR=%.2f dB\n', MSE_l2,    psnr_from_mse
   (MSE_l2));

% --------------------------
% Show results
% --------------------------
figure('Name','Denoising results','NumberTitle','off');
subplot(2,2,1); imshow(Inoisy); title('Noisy');
subplot(2,2,2); imshow(x_tv);   title(sprintf('TV (\\lambda=%.3f)',
    lambda_tv));
subplot(2,2,3); imshow(x_l1);   title(sprintf('L1 (\\lambda=%.3f)',
    lambda_l1));
subplot(2,2,4); imshow(x_l2);   title(sprintf('L2 (\\lambda=%.3f)',
```

```
    lambda_l2));
```

Listing 5 – Affichage des résultats

Les résultats pour cette dernière section sont les suivantes :

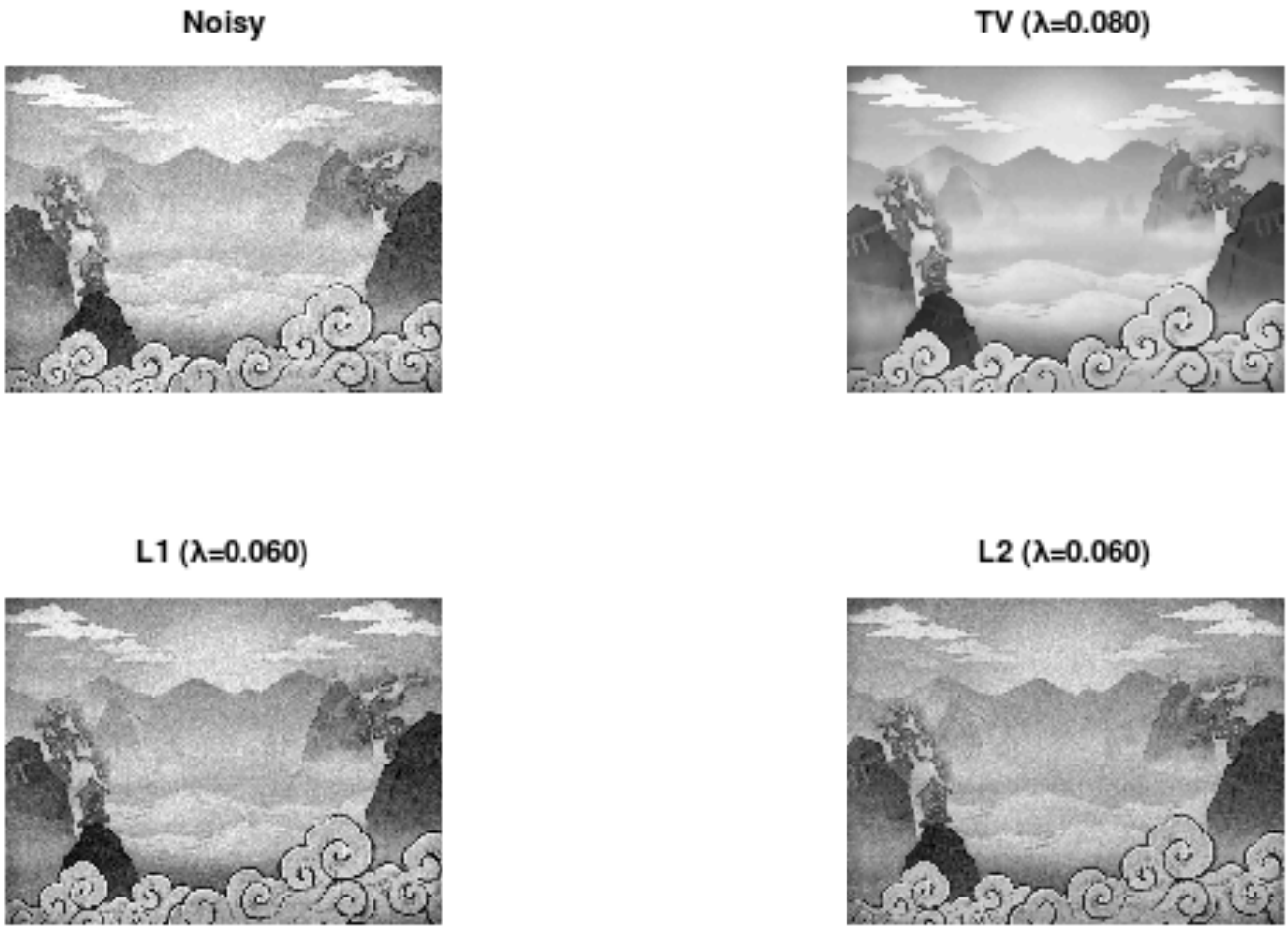

FIGURE 2 – Résultat obtenu après optimisation avec Unlockbox

```
--- MSE / PSNR ---
Noisy: MSE=0.002465, PSNR=26.08 dB
TV   : MSE=0.000472, PSNR=33.26 dB
L1   : MSE=0.006098, PSNR=22.15 dB
L2   : MSE=0.007625, PSNR=21.18 dB
>>
```

FIGURE 3 – Valeur après minimisation

On peut constater que dans notre cas, la meilleure approxmation utilisée est l'approximation TV dont le résultat se rapproche le plus de l'image en niveau de gris initial.

Il est donc judicieux pour un problème donné, de tester plusieurs méthodes d'optimisation pour voir quel résultat est le meilleur.

5 Conclusion

UNLocBoX offre une approche élégante et modulaire pour traiter les problèmes d'optimisation convexe en Octave ou MATLAB. Grâce à ses opérateurs proximaux et à ses solveurs génériques, il permet de comparer rapidement plusieurs régularisations convexes et de sélectionner celle offrant le meilleur compromis entre fidélité et lissage.