Tutoriel : Débruitage avec UnLocBox (Pénalisation L1 ou L2)

Introduction

Bienvenue dans ce tutoriel interactif et visuel sur le débruitage de signaux et d'images en utilisant la **toolbox UnLocBox** sous MATLAB ! Nous allons explorer étape par étape comment formuler et résoudre un problème de régression/débruitage avec une pénalisation **L1** (pour favoriser la sparsité, comme dans Lasso) ou **L2** (pour une régularisation plus lisse, comme Tikhonov).

Objectif principal : Vous guider de manière claire et pratique, en expliquant les principes, en définissant les fonctions clés (différentiables et proximales), en choisissant un solveur adapté, et en appelant `solvep`. Chaque ligne de code sera commentée pour une compréhension immédiate. Pour rendre cela vivant, nous inclurons des **exemples visuels** (avant/après) sur un signal 1D et une image 2D.

Prêt ? Allons-y !

1 Principe du problème

Imaginez que vous avez un signal ou une image originale x (un vecteur), mais vous n'observez que des données bruitées y, via un opérateur linéaire A (qui pourrait être l'identité pour un simple débruitage, ou représenter un flou, un sous-échantillonnage, etc.).

Le problème de débruitage régularisé se formule comme une minimisation :

$$\min_x \frac{1}{2}\|Ax - y\|_2^2 + \lambda \cdot R(x)$$

- **Le terme de fidélité** $\frac{1}{2}\|Ax-y\|_2^2$ mesure l'erreur quadratique entre les prédictions et les observations. Il est **différentiable**, ce qui nous permet d'utiliser son gradient pour l'optimisation.
- **Le terme de régularisation** $R(x)$ contrôle la complexité de x :
  - Pour **L1** (sparsité) : $R(x) = \|x\|_1$ Favorise des solutions avec beaucoup de zéros (idéal pour les signaux compressibles).
  - Pour **L2** (lissage) : $R(x) = \frac{1}{2}\|x\|_2^2$ Pénalise les grandes valeurs, rendant la solution plus "lisse".
- λ est le paramètre qui équilibre fidélité aux données et régularisation. Trop petit ? Trop de bruit reste. Trop grand ? Tout devient trop simplifié !

1.1 Approche algorithmique

Nous utilisons des **algorithmes proximaux** comme Forward-Backward (ISTA) ou son version accélérée FISTA. L'idée :

- **Forward** : Pas de gradient sur le terme différentiable (comme l'erreur quadratique).
- **Backward** : Opérateur proximal sur le terme non-différentiable (comme L1, résolu par soft-thresholding).

UnLocBox simplifie tout ça avec des structures **F** (différentiable) et **G** (proximable), et la fonction `solvep` pour assembler et exécuter le solveur. C'est puissant et modulaire !

2 Définition des fonctions pour UnLocBox

UnLocBox adore les structures fonctionnelles. Définissons **F** et **G** pour chaque cas. (Astuce : Utilisez des fonctions anonymes pour plus de flexibilité !)

2.1 Cas L1 (Pénalisation non-différentiable)

- **F** : Le terme différentiable $\frac{1}{2}\|Ax - y\|_2^2$.
- **G** : Le terme proximable $\lambda\|x\|_1$, résolu par soft-thresholding.

Exemple MATLAB commenté ligne par ligne :

```matlab
% --- Données d'exemple ---
% A : matrice ou handle d'opérateur linéaire.
% y : observations (vecteur ou matrice vectorisée).
% lambda : paramètre de régularisation.

% 1) Fonction F (terme différentiable : fidélité aux données)
F.eval = @(x) 0.5 * norm(A * x - y(:))^2;   % Calcule la valeur scalaire de F(x)  l'erreur quadratique.
F.grad = @(x) A' * (A * x - y(:));         % Fournit le gradient de F en x  pour les pas de descente.

% 2) Fonction G (terme L1 non-différentiable, mais proximable)
G.eval = @(x) lambda * norm(x(:), 1);      % Calcule la valeur de G(x)  la norme L1 pondérée.
G.prox = @(x, T) sign(x) .* max(abs(x) - T * lambda, 0);  % Applique l'opérateur proximal : soft-thresholding avec seuil T*lambda.

% 3) Paramètres du solveur (personnalisez pour votre problème !)
param.verbose = 1;     % Affiche des infos pendant l'exécution (0 pour silencieux).
param.maxit = 200;     % Nombre maximal d'itérations  arrêtez tôt si ça converge vite.
param.tol = 1e-6;      % Tolérance sur la convergence (plus petit = plus précis, mais plus long).
param.beta = 1;        % Pas initial (optionnel)  certains solveurs l'ajustent automatiquement.

```

```
% 4) Appel du solveur : Choisissons 'fista' pour l'accélération Nesterov !
[sol, infos] = solvep(F, G, 'fista', param);
% sol : La solution estimée (vecteur x optimisé).
% infos : Structure avec historique (coûts, erreurs, itérations) super pour le debugging !
```

Remarques :

— Si A est grand ou structuré (ex. : convolution), utilisez des handles comme `A = @(x) conv(x, kernel);` et `A' = @(y) conv(y, flip(kernel));` pour éviter les matrices denses.

— Le proximal de G est $_{T\lambda\|\cdot\|_1}(x) = (x) \cdot \max(|x| - T\lambda, 0)$. C'est rapide et vectorisé !

2.2 Cas L2 (Pénalisation différentiable)

Ici, tout est différentiable ! On peut tout mettre dans **F** et utiliser un solveur de gradient simple.

Exemple MATLAB :

```
% F : Tout le coût (fidélité + régularisation L2)
F.eval = @(x) 0.5 * norm(A * x - y(:))^2 + 0.5 * lambda * norm(x(:))^2;  % Valeur totale : erreur + (1/2) lambda ||x||^2.
F.grad = @(x) A' * (A * x - y(:)) + lambda * x(:);  % Gradient combiné.

% Pas besoin de G (laissez vide !)
param.maxit = 200; param.tol = 1e-8; param.verbose = 1;

% Appel : Utilisez 'gradient_descent' pour la simplicité.
[sol, infos] = solvep(F, [], 'gradient_descent', param);
```

Alternative : Séparez en F (fidélité) et G (L2), mais fournissez `G.grad` au lieu de `G.prox` si le solveur le permet. Pour L2, c'est souvent plus simple de tout fusionner.

3 Choix du solveur

Choisir le bon solveur, c'est comme sélectionner une voiture : ça dépend du terrain !

— **Forward-Backward / ISTA** : Simple et robuste, mais lent pour les grands problèmes.

— **FISTA** : Accéléré (merci Nesterov !) Idéal pour L1 + quadratique. Vitesse boostée sans perte de stabilité.

— **ADMM** : Parfait pour des décompositions complexes (ex. : contraintes multiples).

Pour notre débruitage L1/L2, **FISTA** est le champion : rapide et efficace. Utilisez-le par défaut !

4 Exemple concret 1 : Débruitage d'un signal 1D

Testons sur un signal sinusoïdal bruité. Visualisez l'avant/après pour voir la magie !

```
% Exemple : Denoising d'un signal sinusoïdal corrompu par bruit gaussien
n = 512;                          % Taille du signal  ajustez pour tester.
x_true = sin(2*pi*(0:n-1)/n)';    % Signal propre (sinusoïde parfaite).
sigma = 0.3;                      % Niveau de bruit  plus grand = plus challenging !
y = x_true + sigma * randn(n,1);  % Observations bruitées.
A = eye(n);                       % Opérateur identité (simple débruitage).
lambda = 0.1;                     % Régularisation L1  testez des valeurs !

% Définitions F et G
F.eval = @(x) 0.5 * norm(A * x - y)^2;
F.grad = @(x) A' * (A * x - y);
G.eval = @(x) lambda * norm(x(:), 1);
G.prox = @(x, T) sign(x) .* max(abs(x) - T * lambda, 0);

param.verbose = 1; param.maxit = 200; param.tol = 1e-6;
[sol, infos] = solvep(F, G, 'fista', param);

% Affichage visuel : Avant / Après
figure;
subplot(2,1,1); plot(y); title('Signal Bruité (y)'); ylabel('Amplitude');
subplot(2,1,2); plot(sol); hold on; plot(x_true, '--r'); title('Solution Après L1 (sol) et Signal Vrai (---)'); legend('Estimé', 'Vrai'); ylabel('Amplitude'); xlabel('Échantillons');
```

Résultat attendu : Le signal bruité (haut) est chaotique ; après débruitage (bas), il se rapproche de la sinusoïde pure. Essayez avec L2 pour comparer !

5 Exemple concret 2 : Débruitage d'une image 2D

Passons aux images ! Utilisons `cameraman.tif` (fourni par MATLAB). La L1 favorise la sparsité des pixels.

```
% Lecture image et ajout de bruit
I = im2double(imread('cameraman.tif'));  % Image grayscale en [0,1].
[m, n] = size(I);
y = I + 0.05 * randn(m, n);              % Image bruitée (bruit gaussien).
yv = y(:);                               % Vectorisation pour UnLocBox.
```

```
A = @(x) x;                              % Identité (pas de flou ici).
lambda = 0.08;                           % Ajustez pour un bon équilibre.

% Définitions F et G
F.eval = @(x) 0.5 * norm(x - yv)^2;
F.grad = @(x) x - yv;
G.eval = @(x) lambda * norm(x(:), 1);
G.prox = @(x, T) sign(x) .* max(abs(x) - T * lambda, 0);

param.verbose = 1; param.maxit = 300; param.tol = 1e-6;
[solv, infos] = solvep(F, G, 'fista', param);
I_denoised = reshape(solv, m, n);        % Remise en forme 2D.

% Affichage côte à côte
figure('Position', [100 100 1200 400]);  % Grande fenêtre pour mieux voir !
subplot(1,3,1); imshow(I); title('Originale');
subplot(1,3,2); imshow(y); title('Bruitée');
subplot(1,3,3); imshow(I_denoised); title('Débruitée (L1)');
```

Variante avancée : Pour de meilleurs résultats, appliquez L1 dans un domaine transformé (ex. : ondelettes). Définissez `G.prox = @(x, T) iwt(soft`$_threshold(wt(x), T * lambda));$ `owt` et `iwt` $sontvostransformsd'ondelettes.arendl'imageplusnette$!

Résultat visuel : L'original est clair ; la bruitée est granuleuse ; la débruitée est lisse tout en préservant les détails.

6 Comment choisir λ et surveiller la convergence

λ est clé :

- Petit : Résultat proche de y (bruit persistant).

- Grand : Solution trop "zéro" ou lisse.

Astuces :

- Testez plusieurs valeurs et inspectez visuellement.

- Heuristiques : Pour sparsité, $\lambda \approx \sigma\sqrt{2 \log N}$ (N = taille de x).

- Validation croisée pour les pros.

Utilisez infos pour monitorer :

```
figure; plot([infos.obj]); title('Historique de la Fonction Objectif'); xlabel('Itérations'); ylabel('Coût');
```

Une courbe descendante = convergence heureuse ! Si ça stagne, augmentez maxit ou ajustez le pas.

7 Conseils pratiques et dépannage

- **Lent ou divergent ?** Vérifiez le pas : Estimez la constante Lipschitz (ex. : L = norm(A)$^2; param.gamma = 1/L;$).**Grandes données ?**$Utilisezdeshandlespour$ `A` $etvitezle$

- **Prox complexes ?** Testez-les isolément (ex. : TV pour les images : utilisez prox$_t vd'UnLocBox)$.**Normalisation** : $Toujoursscalezvosdonnes(ex. : imagesen[0,1])pourunec$
- **Bonus** : Ajoutez des contraintes (ex. : positivité) via des prox projecteurs.

Si ça coince, consultez la doc UnLocBox ou testez sur de petits exemples.

8 Script complet prêt à exécuter

Copiez-collez ça dans MATLAB et lancez ! (Adaptez lambda pour votre image.)

```
% Tuto complet : Débruitage L1 d'une image
clear; close all; clc;
I = im2double(imread('cameraman.tif'));
[m, n] = size(I);
y = I + 0.05 * randn(m, n);  % Bruit gaussien.
yv = y(:);
lambda = 0.08;               % Ajustez-moi !

% F (fidélité)
F.eval = @(x) 0.5 * norm(x - yv)^2;
F.grad = @(x) x - yv;

% G (L1)
G.eval = @(x) lambda * norm(x(:), 1);
G.prox = @(x, T) sign(x) .* max(abs(x) - T * lambda, 0);

% Paramètres
param.verbose = 1;
param.maxit = 300;
param.tol = 1e-6;

% Solveur
[solv, infos] = solvep(F, G, 'fista', param);
I_denoised = reshape(solv, m, n);

% Affichage
figure('Position', [100 100 1200 400]);
subplot(1,3,1); imshow(I); title('Originale');
subplot(1,3,2); imshow(y); title('Bruitée');
subplot(1,3,3); imshow(I_denoised); title('Débruitée (L1)');

% Historique
figure; plot([infos.obj]); title('Historique de la Fonction
   Objectif'); xlabel('Itérations'); ylabel('Coût');
```

Fin du tuto ! Vous êtes maintenant un pro du débruitage avec UnLocBox. Essayez avec vos propres données et partagez vos résultats. Des questions ? Testez et itérez !