INF 4127 : Optimisation II
TPE02 : UNLocBoX - Boîte à outils MATLAB pour l'optimisation convexe

Méthodes de splitting proximal

22Y058 - MVOGO MONDOMAN Franck Stéphane
22Y567 - ETOUNDI TSANGA ELIHU F.
21T2487 -ABANDA ARMAND WILFRIED

Université de Yaoundé I
Département d'Informatique

Supervisé par le **Pr PAULIN MELATAGIA**

Plan de la présentation

Pourquoi l'optimisation convexe ?

Il s'agit de résoudre le problème d'optimisation suivant :

$$\min_{x} f(x), \quad \text{où } f \text{ est une fonction convexe.}$$

Avantages :

- Garantie de convergence vers l'optimum global
- Outil fondamental pour résoudre des problèmes complexes de manière efficace et garantie.

Applications :

- Traitement d'images
- Traitement du signal
- Compression de données
- Machine learning

Motivation

Des outils numériques existent pour faciliter l'optimisation convexe : CVX, TFOCS, LTFAT
Parmi ces outils **UNLocBoX** fera l'objet de notre étude.

Qu'est-ce que UNLocBoX ?

Définition

UNLocBoX est une boîte à outils MATLAB conçue pour résoudre des problèmes d'optimisation convexe de la forme :

$$\min_{x \in \mathbb{R}^N} \sum_{n=1}^{K} f_n(x)$$

- Développé par l'EPFL (École Polytechnique Fédérale de Lausanne)
- Basé sur les méthodes de **splitting proximal**
- Adapté aux problèmes de grande dimension (Big Data)
- Open source et bien documenté

Splitting proximal : idée de base

Principe

Le **splitting proximal** consiste à résoudre un problème d'optimisation en **découpant la fonction objective** :

$$f(x) = f_1(x) + f_2(x)$$

- f_1 : différentiable $\to$ on peut utiliser le gradient
- f_2 : différentiable ou non $\to$ si non, on utilisera un opérateur proximal

Exemple : régression linéaire simple avec bruit gaussien

$$f(x) = \underbrace{\frac{1}{2}\|Ax - y\|_2^2}_{f_1(x)} + \underbrace{\gamma\|x\|_2^2}_{f_2(x)}$$

Les 02 fonctions sont différentiables **Itération pratique (splitting)** :

$$x_k \leftarrow x_{k-1} - \lambda \nabla f_1(x_{k-1}), \quad x_k \leftarrow x_k - \lambda \nabla f_2(x_k)$$

Opérateur proximal : gérer le non différentiable

Definition

Permet de traiter des fonctions non différentiables

Pour une fonction convexe $f : \mathbb{R}^N \to (-\infty, +\infty]$, l'opérateur proximal est :

$$\mathrm{prox}_f(x) := \arg\min_{y \in \mathbb{R}^N} \left\{ \frac{1}{2}\|x - y\|_2^2 + f(y) \right\}$$

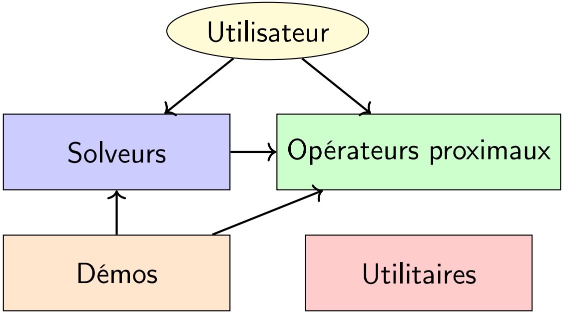
Cas pratique : norme ℓ_1 (sparsité)

$$f_2(x) = \gamma \|x\|_1$$

Forward-backward (splitting proximal) :

$$x_{\text{temp}} = x_k - t\nabla f_1(x_k), \quad x_{k+1} = \mathrm{prox}_{tf_2}(x_{\text{temp}}) = \begin{cases} x_{\text{temp},i} - t\gamma & \text{si } x_{\text{temp},i} > t\gamma \\ 0 & \text{si } |x_{\text{temp},i}| \leq t\gamma \\ x_{\text{temp},i} + t\gamma & \text{si } x_{\text{temp},i} < -t\gamma \end{cases}$$

Architecture générale



Composants principaux :

- **Solveurs** : FISTA, Douglas-Rachford, ADMM, Chambolle-Pock...
- **Opérateurs proximaux** : ℓ_1, ℓ_2, TV, nucléaire...

Définir une fonction différentiable

Structure d'une fonction

Chaque fonction $f_k(x)$ est modélisée par une structure MATLAB avec les champs :

- `eval` : évalue $f(x)$
- `grad` : calcule $\nabla f(x)$ (si différentiable)
- `prox` : calcule $\text{prox}_{\gamma f}(x)$ (si non-différentiable)
- `beta` : constante de Lipschitz du gradient

```
% Exemple : f(x) = 5||Ax - y||_2^2
f.eval = @(x) 5 * norm(A*x - y)^2;
f.grad = @(x) 10 * A'*(A*x - y);
f.beta = 10 * norm(A)^2;
```

Définir une fonction non-différentiable

```
% Exemple : f(x) = lambda * ||x||_1
lambda = 7;
f.eval = @(x) lambda * norm(x, 1);
f.prox = @(x, T) prox_l1(x, lambda*T);
```

Opérateurs proximaux disponibles

- `prox_l1` : norme ℓ_1 (parcimonie)
- `prox_l2` : norme ℓ_2 (régularisation)
- `prox_tv` : variation totale (images)
- `prox_nuclearnorm` : norme nucléaire (matrices low-rank)
- `proj_b2` : projection sur boule ℓ_2 (contrainte)

Contraintes sur les fonctions non-differentiable

- Un prox peut avoir trois paramètres :

```
f.prox = @(x,T) prox\_l1(x, lambda*T, param);}
```

où x = signal courant, lambda = poids de la fonction objective, param = structure optionnelle contenant par exemple A, At, tight, nu.

- Pour restreindre les solutions à un ensemble convexe $C \subset \mathbb{R}^L$, on utilise un opérateur de projection P_C au lieu de l'opérateur proximal classique. Définition :

$$P_C(y) = \arg\min_{x \in C} \|y - x\|_2^2$$

Exemple : contrainte $\|x\|_2 \leq 2$

```
f.eval = @(x) eps;
param_b2.epsilon = 2;
f.prox = @(x,T) prox_b2(x,T,param_b2);
```

Introduction aux solveurs UNLocBoX

- Un **solveur** est un algorithme qui cherche à trouver le minimum (ou le maximum) d'une fonction ou d'une somme de fonctions.
- Les solveurs UNLocBoX minimisent des fonctions différentiables et/ou non-différentiables.
- **Usage** : un solveur prend trois entrées principales : le point initial x_0, les fonctions à minimiser, et une structure optionnelle de paramètres ‘param‘.
- Sélection automatique d'un solveur : **solvep** (Pour débutant)

Exemple d'appel de solveur : Selection automatique.

```
sol = solvep(x0, {f1, f2, f3}, param);
```

Panorama des solveurs UNLocBoX

Il existe 02 grandes catégories de solveurs

- Solveurs spécifiques : 02 fonctions max.
- Solveurs généraux ; K fonctions.

Solveur	**Type de fonctions**	**Vitesse**
Forward-Backward (FISTA)	1 diff. + 1 non-diff.	Rapide
Douglas-Rachford	2 non-diff.	Moyen
ADMM	2 non-diff. + contrainte	Moyen
Chambolle-Pock	2 non-diff. + 1 diff.	Rapide
PPXA	K non-diff.	Lent
SDMM	K non-diff.	Moyen

```
sol = forward_backward(x0, f1, f2, param);
```

Quelques solveurs : Forward-Backward (FISTA)

Problème résolu

$$\min_x f_1(x) + f_2(x)$$

où f_1 est différentiable et f_2 est non-différentiable.
Très rapide en pratique - Convergence $O(1/k^2)$ avec FISTA

Algorithme :

1. Étape gradient : $z_i = x_{i-1} - \gamma \nabla f_1(x_{i-1})$
2. Étape proximale : $x_i = \mathrm{prox}_{\gamma f_2}(z_i)$
3. Accélération de Nesterov (FISTA)

MATLAB

```
x0 = randn(n,1);
param.verbose = 1;
[x, info] = forward_backward(x0, f1, f2, param);
```

Problème résolu

$$\min_x f_1(x) + f_2(x)$$

où f_1 et f_2 sont toutes deux non-différentiables.
Applications : Problèmes avec contraintes - Débruitage d'images avec inpainting

Algorithme :

1. $y_i = \text{prox}_{\gamma f_1}(x_{i-1})$
2. $z_i = \text{prox}_{\gamma f_2}(2y_i - x_{i-1})$
3. $x_i = x_{i-1} + \lambda(z_i - y_i)$

MATLAB

```
x0 = randn(n,1);
param.verbose = 1;
[x, info] = douglas_rachford(x0, f1, f2, param);
```

Cas pratique : Formulation du problème

Objectif : reconstruire un signal x à partir de mesures bruitées $y = Ax + n$ en imposant une **régularité L1** (parcimonie).

$$\min_x \|Ax - y\|_2^2 + \lambda \|x\|_1$$

Interprétation

- $\|Ax - y\|_2^2$: fidélité aux observations
- $\lambda \|x\|_1$: régularisation favorisant la parcimonie
- λ : compromis entre précision et simplicité du signal

Implémentation sous UNLocBoX — Étape 1

```matlab
% --- Donn es synth tiques ---
n = 100;  % dimension du signal inconnu
m = 50;   % nombre de mesures

A = randn(m, n);      % matrice de mesure
x_true = sprandn(n, 1, 0.1);  % signal parcimonieux (10% non nuls)
y = A * x_true + 0.01*randn(m,1);  % observations bruit es

% --- Fonction 1 : fid lit  aux donn es ---
f1.eval = @(x) norm(A*x - y)^2;
f1.grad = @(x) 2*A'*(A*x - y);
f1.beta = 2*norm(A)^2;  % lipschitz

% --- Fonction 2 : r gularisation L1 ---
lambda = 0.05;
f2.eval = @(x) lambda*norm(x,1);
f2.prox = @(x,T) prox_l1(x, lambda*T);
```

Ajout de paramètres ett résolution

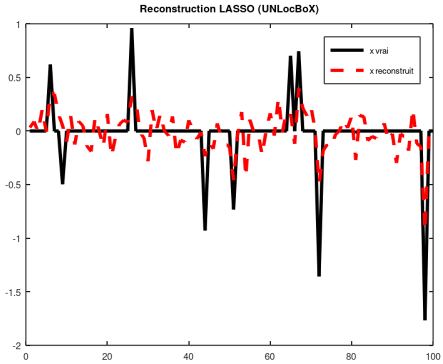
```
% --- Param tres du solveur ---
param.verbose = 2;
param.maxit = 200;
param.tol = 1e-4;
param.gamma = 0.9 / f1.beta;  % PAS STABLE

% --- Point initial ---
x0 = zeros(n,1);

% --- R solution ---
sol = solvep(x0, {f1, f2}, param);

% --- R sultat ---
fprintf('\nErreur␣de␣reconstruction␣:␣%.4f\n', norm(sol - x_true) / norm(
    x_true));
```

Analyse des résultats




```
FORWARD_BACKWARD:
 f(x^*) = 7.184967e-01, rel_eval = 1.382476e-04
200 iterations
Stopping criterion: MAX_IT

Erreur de reconstruction : 0.6512
```

- Valeur finale de la fonction objectif : $f(x^*) = 0.7185$.
- Critère de convergence relatif : $\text{rel\_eval} = 1.38 \times 10^{-4} \rightarrow$ le solveur a presque convergé.
- Nombre d'itérations : 200 (MAX_IT) $\rightarrow$ le solveur a atteint la limite maximale sans atteindre la tolérance souhaitée.
- Erreur de reconstruction : 0.6512 $\rightarrow$ le signal reconstruit diffère encore d'environ 65% du signal original.

Avantages et limites de UNLocBoX

Avantages :

Pour l'utilisateur :

- + Interface simple et intuitive
- + Documentation complète
- + Nombreux exemples
- + Sélection automatique du solveur

Performances :

- + Algorithmes récents et efficaces
- + Scalable (grandes dimensions)
- + Support GPU (limité)
- + Open source

Limites et considérations

- Nécessite une connaissance minimale en optimisation convexe
- Pas complètement "boîte noire" comme CVX
- Certains opérateurs proximaux doivent être définis manuellement
- Support GPU encore en développement

Conclusion

Installation sur MATLAB

1. Télécharger UNLocBoX depuis : `https://lts2.epfl.ch/unlocbox/`
2. Ajouter UNLocBoX au chemin MATLAB : **addpath(genpath('C :/UNLocBoX'))** ;
3. Initialiser UNLocBoX en exécutant : **init_unlocbox** ;
4. Vérifier que tout fonctionne en testant une fonction de base, par exemple : **help solvep**

Resumé

1. UNLocBoX résout des problèmes d'optimisation convexe par splitting proximal
2. Architecture modulaire : solveurs + opérateurs proximaux
3. Scalable et efficace pour le Big Data
4. Applications variées en traitement d'images et apprentissage