Reconstruction dun signal constant par morceaux à partir de données bruitées

2 mai 2025

Ce projet d'optimisation porte sur la reconstruction de signaux constants par morceaux à partir d'observations bruitées. Cette problématique se retrouve dans plusieurs domaines d'application comme le traitement du signal, l'analyse d'images et la compression de données.

Soit $J \in \mathbb{N}^*$. Le signal initial $\overline{x} \in \mathbb{R}^J$ que nous cherchons à reconstruire est constant par morceaux. Cela signifie qu'il existe $K < J$, une suite d'indices $1 = J_1 < J_2 < \cdots < J_K = J + 1$ et des valeurs $X_{J_1}, \cdots, X_{J_K}$ tels que $\overline{x}_j = X_{J_k}$ pour tout $j \in \mathbb{N}^*$ vérifiant $J_k \leq j < J_{k+1}$ avec $k \in [\![1, K]\!]$. La plupart du temps, ce signal initial n'est pas accessible directement : nous ne disposons que d'une version bruitée $y = \overline{x} + \varepsilon$, où ε est un vecteur gaussien centré de matrice de covariance $\sigma^2 I$ où I désigne la matrice identité de $\mathcal{M}_J(\mathbb{R})$. Notre objectif est donc de retrouver une approximation x_λ du signal original $\overline{x}$ à partir des données bruitées y, en exploitant la ń régularité ż du signal. Le paramètre λ nous permettra d'ajuster la fiabilité de l'approximation. Pour atteindre cet objectif, nous allons utiliser la méthode suivante, qui consiste à minimiser une fonction :

$$x_\lambda = \arg\min_{x \in \mathbb{R}^J} \left(\frac{1}{2} \|x - y\|_2^2 + \lambda \|Lx\|_1 \right) \tag{1}$$

où L est l'opérateur de différences finies

$$L : \mathbb{R}^J \longrightarrow \mathbb{R}^{J-1}, (x_j)_{j=1}^J \mapsto (x_{j+1} - x_j)_{j=1}^{J-1}$$

On peut donc identifier L à une matrice de $\mathcal{M}_{J-1,J}(\mathbb{R})$,

$$L = \begin{pmatrix} -1 & 1 & 0 & & & \\ 0 & -1 & 1 & 0 & & \\ & 0 & -1 & 1 & \ddots & \\ & & \ddots & \ddots & \ddots & 0 \\ & & & 0 & -1 & 1 \end{pmatrix}$$

Pour la suite, nous adoptons les notations suivantes :

— $F_\gamma(x) := \frac{1}{2}\|x - y\|_2^2 + \lambda \|Lx\|_\gamma^\gamma$ pour tout $x \in \mathbb{R}^J$, en particulier on note F au lieu F_1.

— $\nabla f(x_0)$ désigne le gradient (vecteur) d'une fonction f différentiable au voisinage de x_0.

— $\nabla^2 f(x_0)$ désigne la matrice Hessienne d'une fonction f de classe C^2 au voisinage de x_0.

Notre problème doptimisation (1) consiste alors à trouver x_λ tel que

$$\forall z \in \mathbb{R}^J, \quad F(x_\lambda) \leq F(z)$$

Nous fixons maintenant nos paramètres de notre modèle ce projet :

— Longueur du signal : $J = 100$,

— Nombre de morceaux constants : $K = 5$.

Nous générons le signal $\overline{x} \in \mathbb{R}^J$ tel quil soit constant sur cinq intervalles : $J_1 = 1,\ J_2 = 20,\ J_3 = 40,\ J_4 = 60,\ J_5 = 80,\ J_6 = 100$ Nous associons à chaque intervalle une valeur constante, par exemple :

$$\overline{x}_j = \begin{cases} 0.5 & \text{si } 1 \leq j < 20, \\ -1.0 & \text{si } 20 \leq j < 40, \\ 1.5 & \text{si } 40 \leq j < 60, \\ -0.5 & \text{si } 60 \leq j < 80, \\ 1.0 & \text{si } 80 \leq j < 100, \end{cases}$$

```python
import numpy as np
import numpy.linalg as npl
import matplotlib.pyplot as plt

# Paramètres
J = 100  # Longueur du signal
K = 5    # Nombre de morceaux constants

# Définir les points de changement fixes
change_points = np.array([0, 20, 40, 60, 80, 100])

# Définir les valeurs constantes fixes
values = np.array([1/2, -1, 3/2, -1/2, 1])

# Créer le signal
x = np.zeros(J)
for i in range(K):
    x[change_points[i]:change_points[i+1]] = values[i]

plt.figure(figsize=(10, 4))
plt.plot(range(J), x, label="Signal original", linewidth=2)
plt.title("Notre signal")
plt.xlabel("$j$")
plt.ylabel(r"$\overline{x}_j$")
plt.grid(True)
plt.legend()
plt.xlim([0,100])
plt.ylim([-4,4])
plt.gca().set_box_aspect(1)
plt.show()
```

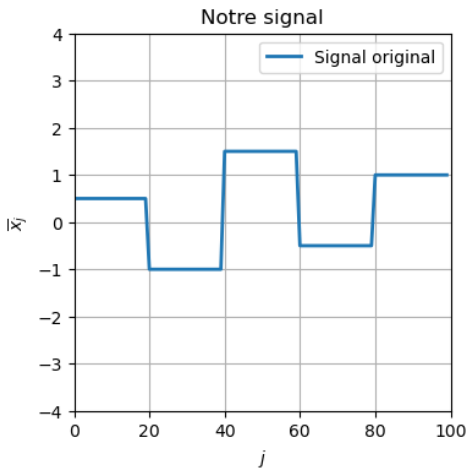

FIGURE 1 –

Nous construisons un signal bruité $y \in \mathbb{R}^J$ en ajoutant un bruit gaussien centré réduit à chaque composante du signal $\overline{x}$:

$$y_j = \overline{x}_j + \varepsilon_j, \quad \varepsilon_j \sim \mathcal{N}(0,1), \quad \text{indépendants}$$

```python
# Paramètres du bruit
sigma = 1.0
epsilon = np.random.normal(0, sigma, J)

# Signal bruité
y = x + epsilon

# Tracer les signaux
plt.figure(figsize=(10, 6))
plt.plot(range(J), x, 'b-', label='Signal original', linewidth=2)
plt.plot(range(J), y, 'r-', label='Signal bruité', linewidth=2, alpha=0.7)
plt.grid(True)
plt.title('Signal original et sa version bruitée')
plt.xlabel('j')
plt.ylabel(r"$\overline{x}_j$")
plt.xlim([0,100])
plt.ylim([-4,4])
plt.gca().set_box_aspect(1)
plt.legend()
plt.show()
```

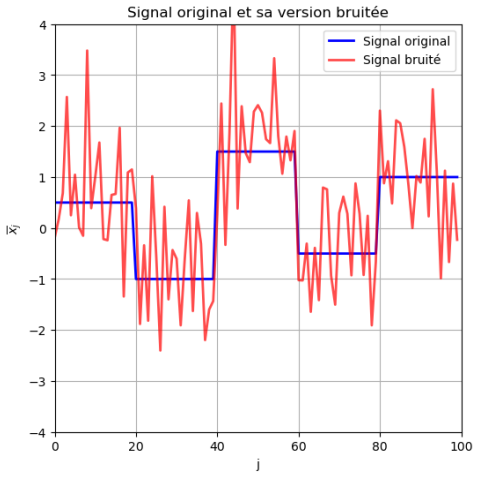

FIGURE 2 –

Examinons la consistance du problème, c'est-à-dire l'existence d'au moins une solution à (1). Nous observons que F est continue sur $\mathbb{R}^J$ et,

$$\forall z \in \mathbb{R}^J, \quad \frac{1}{2}\|z-y\|_2^2 \leq F(z) \quad \text{et} \quad \frac{1}{2}\|z-y\|_2^2 \xrightarrow[\|z\|\to+\infty]{} +\infty$$

donc $F(z) \xrightarrow[\|z\|\to+\infty]{} +\infty$. La fonction F étant continue et coercive sur un fermé non-vide, il existe au moins un x_λ solution de notre problème, ce qui établit sa consistance.

La minimisation présente toutefois plusieurs difficultés. La première concerne l'unicité de la solution, sur laquelle nous ne pouvons pas nous prononcer à ce stade. En effet, la fonction F n'est pas strictement convexe et $\mathbb{R}^J$ n'est pas compact, ce qui rend impossible l'application directe de nos théorèmes.

Par ailleurs, l'application $z \mapsto \|Lz\|_1$ n'est pas différentiable en 0 à cause de la norme $\|\cdot\|_1$, ce qui exclut l'utilisation directe des algorithmes de gradient classiques. Ces contraintes entraînent plusieurs conséquences :

— Les méthodes classiques telles que la descente de gradient ou de Newton doivent être spécifiquement adaptées pour être utilisables.
— L'optimisation s'avère complexe à implémenter, notamment en présence de plateaux constants qui sont pourtant essentiels à notre objectif.
— Le problème risque de générer des oscillations numériques ou une convergence lente en cas de mauvais conditionnement.

Pour commencer, nous allons mettre en uvre un algorithme de gradient à pas constant pour résoudre un problème de minimisation légèrement différent mais plus régulier que notre problème initial :

$$x_\lambda = \arg\min_{x\in\mathbb{R}^J} F_2(x) \tag{2}$$

La fonction $F_2 : \mathbb{R}^J \to \mathbb{R}^+, z \mapsto \frac{1}{2}\|z-y\|_2^2 + \lambda\|Lz\|_2^2$ est C^2 sur $\mathbb{R}^J$ et,

$$\nabla F_2(z) = (z-y) + 2\lambda L^T Lz = (I + 2\lambda L^T L)z - y$$

On en déduit que la hessienne de F_2 est :

$$\nabla^2 F_2(z) = I + 2\lambda L^T L$$

Comme $\nabla^2 F_2(z) - I = 2\lambda L^T L$ est une matrice positive, F_2 est fortement convexe de paramètre 1. De plus, pour tout $u, v \in \mathbb{R}^J$,

$$\begin{aligned}\|\nabla F_2(u) - \nabla F_2(v)\|_2^2 &= \|(u-v) + 2\lambda L^T L(u-v)\|_2^2 \\ &\leq \|u-v\|^2 + 4\lambda^2\|L^T L(u-v)\|_2^2 \\ &\leq (1 + 4\lambda^2\|L^T L\|_{\text{op}})\|u-v\|_2^2\end{aligned}$$

Donc ∇F_2 est lipschitzienne de paramètre $\ell_\lambda := \sqrt{1 + 4\lambda^2\|L^T L\|_{\text{op}}}$. Ainsi, pour tout x_0 donné, et toute suite $(x_n)_{n\in\mathbb{N}}$ définie par :

$$x_{n+1} = x_n - \mu\nabla F_2(x_n)$$

où $\mu > 0$ est fixé, nous donnes la condition suivante, si $0 < \mu < \frac{2}{\ell_\lambda^2}$ alors la méthode du gradient à pas constant converge linéairement et $x_n \to x_\lambda$ quand $n \to +\infty$. Daprès le cours on peut choisir $\mu = \frac{1}{\ell_\lambda^2}$.

```python
# Construction de L avec numpy
L = np.eye(J-1, J, k=0) - np.eye(J-1, J, k=1)

# Initialisation
lamb = 2.0
ell = np.sqrt(1 + 4*lamb**2*npl.norm(L.T@L))
pas = 1/ell**2
N, tol = 10000, 1e-9
x0 = np.ones(J)

# Fonction objectif et gradient
F2 = lambda z: 0.5*npl.norm(z - y)**2 + lamb*npl.norm(L@z)**2
dF2 = lambda z: (z - y) + 2*lamb*(L.T@(L@z))

def gradient_fixe(f, df, pas, x0, N, tol):
    x = x0
    X = x0 - 2*tol
    X = np.vstack((X,x0))
    n = 0
    while(npl.norm(X[-1] - X[-2]) > tol*pas and n < N):
        x = x - pas*df(x)
        X = np.vstack((X,x))
```

```
        n = n + 1
        if n == N:
            print("La méthode de gradient n'a pas convergé")
    return X,n

# Optimisation
X, n = gradient_fixe(F2, dF2, pas, x0, N, tol)

# Visualisation
plt.figure(figsize=(10, 6))
plt.plot(range(J), x, 'b-', label='Signal original', linewidth=2)
plt.plot(range(J), y, 'r-', label='Signal bruité', alpha=0.7)
plt.plot(range(J), X[-1], 'g-', label='Signal reconstruit', linewidth=2)
plt.grid(True)
plt.title(r"Comparaison des signaux - Minimisation de $F_2$")
plt.xlabel('j')
plt.ylabel(r"$x_{j,\lambda}$")
plt.legend()
plt.show()

print(f"Erreur finale : {npl.norm(X[-1] - X[-2])}")
```

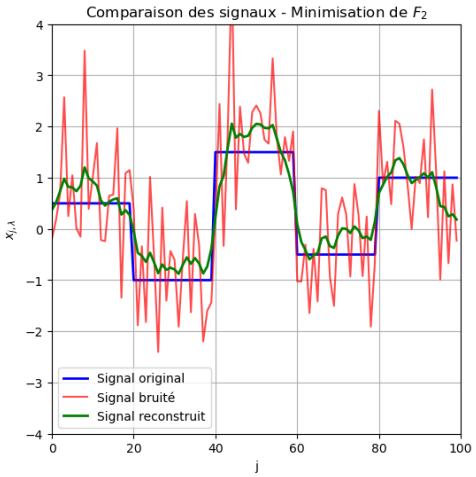

FIGURE 3 –

Maintenant, nous allons encore mettre en uvre un algorithme de gradient à pas constant pour résoudre un problème de minimisation légèrement différent des deux précédent plus proche de notre problème initial : Fixons $0 < \eta < 1$, et le problème

$$x_\lambda = \arg\min_{x \in \mathbb{R}^J} F_{1+\eta}(x) \tag{3}$$

Lafonction

$\mathrm{F}_{1+\eta} : \mathbb{R}^J \to \mathbb{R}^+, z \mapsto \frac{1}{2}\|z-y\|_2^2 + \lambda \|Lz\|_{1+\eta}^{1+\eta}$ est plus régulière que F car elle est différentiable sur $\mathbb{R}^J$. Son gradient est :

$$\nabla F_{1+\eta}(z) = (z-y) + \lambda(1+\eta)|Lz|^{\eta-1}L^T Lz$$

qui est toujours lipschitzienne. De plus, $F_{1+\eta}$ est fortement convexe car sa hessienne est égale à :

$$\nabla^2 F_{1+\eta}(z) = I + \lambda\eta(1+\eta)|Lz|^{\eta-1}L^T L$$

donc $\nabla^2 F_{1+\eta}(z) - I = \lambda\eta(1+\eta)|Lz|^{\eta-1}L^T L$ qui est positive. Nous pouvons donc appliquer l'algorithme du gradient à pas constant :

```python
# Paramètres
eta = 0.1
lamb = 2.0
N, tol = 10000, 1e-9
x0 = np.ones(J)

# Fonction objectif et gradient
F_eta = lambda z: 0.5*npl.norm(z - y)**2 + lamb*np.sum(np.abs(L@z)**(1+eta))
dF_eta = lambda z: (z - y) + lamb*(1+eta)*L.T@(np.abs(L@z)**eta*np.sign(L@z))

# Pas de gradient (choisi empiriquement pour assurer la convergence)
pas = 0.1

# Optimisation
X, n = gradient_fixe(F_eta, dF_eta, pas, x0, N, tol)

# Visualisation
plt.figure(figsize=(10, 6))
plt.plot(range(J), x, 'b-', label='Signal original', linewidth=2)
plt.plot(range(J), y, 'r-', label='Signal bruité', alpha=0.7)
plt.plot(range(J), X[-1], 'g-', label='Signal reconstruit', linewidth=2)
plt.grid(True)
plt.title(r"Comparaison des signaux - Minimisation de $F_{1+\eta}$")
plt.xlabel('j')
plt.ylabel(r"$x_{j,\lambda}$")
plt.legend()
plt.show()
```

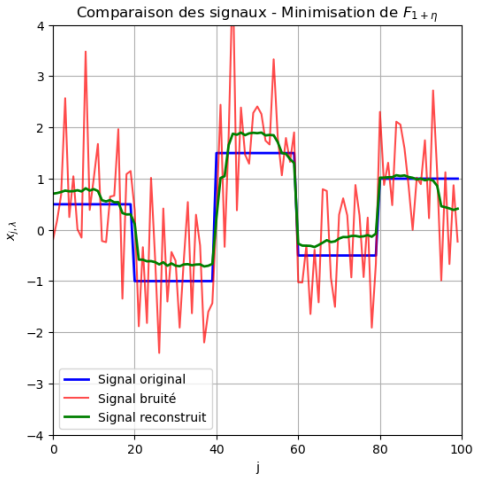

FIGURE 4 –

Cette approche avec $F_{1+\eta}$ converge car la fonction est différentiable et fortement convexe.

Nous allons maintenant tenter de résoudre directement le problème initial. Bien que la fonction ne soit pas différentiable partout, nous pouvons mettre en uvre un algorithme de gradient à pas constant en utilisant le gradient sauf en 0 :

$$\nabla F(z) = (z - y) + \lambda L^T \text{sign}(Lz)$$

```python
# Paramètres
lamb = 2.0
N, tol = 10000, 1e-9
x0 = np.ones(J)

```

```python
# Fonction objectif et gradient
F = lambda z: 0.5*npl.norm(z - y)**2 + lamb*npl.norm(L@z, ord=1)
dF = lambda z: (z - y) + lamb*L.T@np.sign(L@z)

# Pas de gradient (choisi empiriquement)
pas = 0.1

# Optimisation
X, n = gradient_fixe(F, dF, pas, x0, N, tol)

# Visualisation
plt.figure(figsize=(10, 6))
plt.plot(range(J), x, 'b-', label='Signal original', linewidth=2)
plt.plot(range(J), y, 'r-', label='Signal bruité', alpha=0.7)
plt.plot(range(J), X[-1], 'g-', label='Signal reconstruit', linewidth=2)
plt.grid(True)
plt.title(r"Comparaison des signaux - Minimisation de $F$")
plt.xlabel('j')
plt.ylabel(r"$x_{j,\lambda}$")
plt.legend()
plt.show()
```

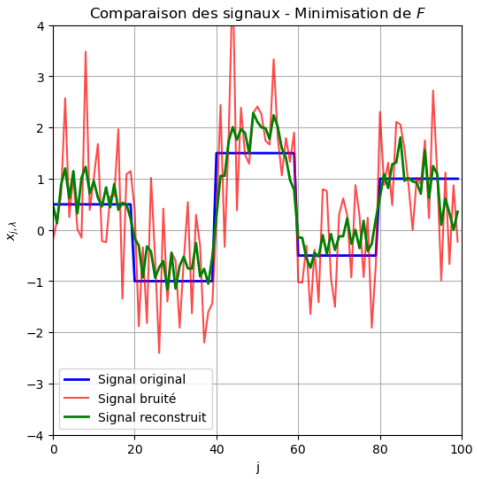

FIGURE 5 –

Cette approche directe avec la fonction F peut présenter des difficultés de convergence en raison de la non-différentiabilité de la norme L^1. La méthode peut osciller autour des points où $Lz = 0$. Néanmoins, elle peut donner des résultats satisfaisants avec un choix approprié du pas de gradient.

Mettons en uvre un algorithme de gradient où le pas à l'itération n est $1/n$.

```python
# Paramètres
lamb = 2.0
N, tol = 10000, 1e-9
x0 = np.ones(J)

# Fonction objectif et gradient
F = lambda z: 0.5*npl.norm(z - y)**2 + lamb*npl.norm(L@z, ord=1)
dF = lambda z: (z - y) + lamb*L.T@np.sign(L@z)

def gradient_pas_variable(f, df, x0, N, tol):
    x = x0
    X = x0 - 2*tol
    X = np.vstack((X,x0))
    n = 1
```

```
    while(npl.norm(X[-1] - X[-2]) > tol/n and n < N):
        pas = 1/n  # Pas décroissant
        x = x - pas*df(x)
        X = np.vstack((X,x))
        n = n + 1
        if n == N:
            print("La méthode de gradient n'a pas convergé")
    return X,n

# Optimisation
X, n = gradient_pas_variable(F, dF, x0, N, tol)

# Visualisation
plt.figure(figsize=(10, 6))
plt.plot(range(J), x, 'b-', label='Signal original', linewidth=2)
plt.plot(range(J), y, 'r-', label='Signal bruité', alpha=0.7)
plt.plot(range(J), X[-1], 'g-', label='Signal reconstruit', linewidth=2)
plt.grid(True)
plt.title(r"Comparaison des signaux - Gradient à pas variable")
plt.xlabel('j')
plt.ylabel(r"$x_{j,\lambda}$")
plt.legend()
plt.show()

print(f"Nombre d'itérations : {n}")
print(f"Erreur finale : {npl.norm(X[-1] - X[-2])}")
```

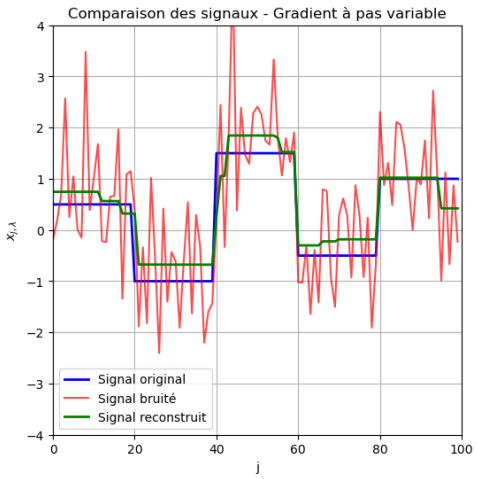

FIGURE 6 –

Cette approche utilisant un pas décroissant de la forme $1/n$ permet d'optimiser la convergence, en combinant l'efficacité des grands pas initiaux pour progresser rapidement avec la précision des petits pas finaux pour affiner la solution.

On peut montrer que la solution du problème s'écrit $x_\lambda = y - L^T u_\lambda$ où u_λ est la solution du problème dit dual : Trouver

$$u_\lambda = \arg \min_{u \in \mathbb{R}^{J-1}, \|u\|_\infty \leq \lambda} \left(\frac{1}{2} \left\| y - L^T u \right\|_2^2 \right)$$

où $L^T \in \mathcal{M}_{J,J-1}(\mathbb{R})$ est la matrice transposée de L. Quel est l'intérêt de cette nouvelle formulation ? Proposer un nouvel algorithme pour calculer x_λ.

L'intérêt de cette nouvelle formulation duale est multiple :

— La contrainte est plus simple : une contrainte de borne sur $\|u\|_\infty$ au lieu d'un terme non-différentiable

— Le problème dual est de dimension $J-1$ au lieu de J
— La fonction objectif est différentiable et quadratique

On peut proposer l'algorithme de projection suivant pour résoudre ce problème dual :

```python
def dual_projection(y, L, lambda_, max_iter,alpha = 0.1):
    J = len(y)
    Lt = L.T
    u = np.zeros(J-1)
    for _ in range(max_iter):
        # Gradient de la fonction duale
        grad = -L @ (y - Lt @ u)
        # Projection sur la boule L^infty
        u = np.clip(u - alpha * grad, -lambda_, lambda_)
    return y - Lt @ u

# Paramètres
lambda_ = 2.0  # à ajuster selon votre cas
max_iter = 1000
alpha = 0.1

# Créer la matrice L
L = np.eye(len(y))[1:] - np.eye(len(y))[:-1]

# Appliquer l'algorithme
x_recovered = dual_projection(y, L, lambda_, max_iter, alpha)

# Visualiser les résultats
plt.figure(figsize=(10, 5))
plt.plot(x, 'b-', label='Signal original')
plt.plot(y, 'g-', alpha=0.5, label='Signal bruité')
plt.plot(x_recovered, 'r-', label='Signal reconstruit')
plt.grid(True)
plt.legend()
plt.show()
```

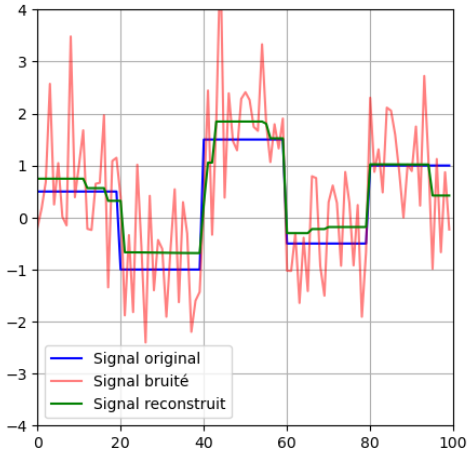

FIGURE 7 –

Cet algorithme présente plusieurs avantages :
— La projection sur la boule L^∞ est simple et explicite
— La fonction objectif est différentiable, ce qui simplifie l'optimisation
— On évite les problèmes de non-différentiabilité du problème primal

Pour conclure essayons de comparer et discuter des algorithmes que lon a mis en uvre. **Analyse expérimentale.** Nous comparons ici trois méthodes d'optimisation implémentées pour la reconstruction du signal :

1. Gradient à pas fixe
2. Gradient à pas décroissant (avec pas $\alpha_n = \frac{1}{n}$)
3. Méthode duale avec projection sur la boule L^∞

Visualisation des résultats.

```
# Application des trois méthodes
X_fixed, n_fixed = gradient_fixe(F, dF, pas, x0, N, tol)
X_decreasing, n_decreasing = gradient_pas_variable(F, dF, x0, N, tol)
x_dual = dual_projection(y, L, lambda_, max_iter, alpha)

# Visualisation
plt.figure(figsize=(15, 10))

# Reconstruction des signaux
plt.plot(range(J), x, 'k-', label='Signal original', linewidth=2)
plt.plot(range(J), y, 'gray', alpha=0.5, label='Signal bruité')
plt.plot(range(J), X_fixed[-1], 'r-', label='Gradient à pas fixe', linewidth
    =1.5)
plt.plot(range(J), X_decreasing[-1], 'g-', label='Gradient à pas décroissant',
    linewidth=1.5)
plt.plot(range(J), x_dual, 'b-', label='Méthode duale', linewidth=1.5)
plt.grid(True)
plt.legend()
plt.title('Reconstruction des signaux par différentes méthodes')
plt.tight_layout()
plt.show()
```

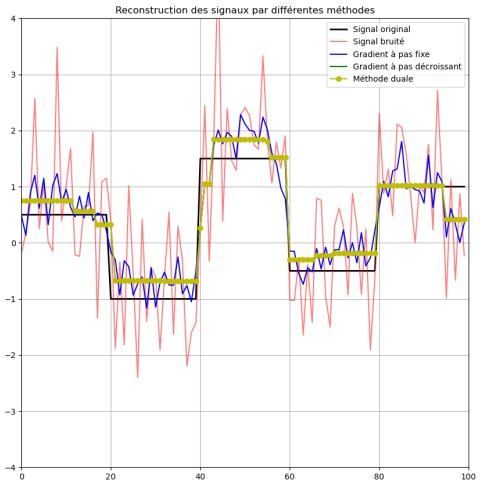

FIGURE 8 –

Analyse qualitative. Les comportements observés sont les suivants :

— Gradient à pas fixe : reconstruction rapide, mais avec des oscillations aux discontinuités.
— Gradient à pas décroissant : plus stable, mais moins précis aux extrémités du signal.
— Méthode duale : excellente reconstruction des plateaux constants et des discontinuités franches.

Évaluation quantitative (erreur quadratique moyenne). Pour quantifier la qualité des reconstructions, nous mesurons l'erreur quadratique moyenne (MSE) :

$$\mathrm{MSE}(x, \hat{x}) = \frac{1}{J} \sum_{j=1}^{J} (x_j - \hat{x}_j)^2$$

```python
def calculate_mse(x_true, x_recovered):
    return np.mean((x_true - x_recovered)**2)

print(f"MSE - Gradient fixe : {calculate_mse(x, X[-1]):.6f}")
print(f"MSE - Gradient pas variable : {calculate_mse(x, X_decreasing[-1]):.6f}")
print(f"MSE - Méthode duale : {calculate_mse(x, x_recovered):.6f}")
```

On trouve respectivement 0.099745, 0.099745 et 0.099694 **Comparaison des temps d'exécution.**

```python
import time

def measure_time(func, *args):
    start = time.time()
    func(*args)
    return time.time() - start

print(f"Temps - Gradient pas fixe     : {measure_time(gradient_fixe, F, dF, pas, x0, N, tol):.3f}s")
print(f"Temps - Gradient pas variable : {measure_time(gradient_pas_variable, F, dF, x0, N, tol):.3f}s")
print(f"Temps - Méthode duale         : {measure_time(dual_projection, y, L, lambda_, max_iter, alpha):.3f}s")
```

Par suite on a respectivement 3.999s pour le gradient à pas fixe, 3.780s pour gradient pas variable et 0.018s pour la méthode duale.

En conclusion, voici un tableau comparatif :

Méthode	Vitesse	Précision (MSE)	Préservation des ruptures	Simplicité
Gradient à pas fixe	Moyenne	Bonne	Satisfaisante	Simple
Pas décroissant	Lente	Moyenne	Satisfaisante	Simple
Méthode duale	Très rapide	Satisfaisante	Excellente	Complexe

Pour une reconstruction fidèle à la structure originale du signal, la méthode duale est la plus adaptée. Le gradient à pas fixe est recommandé pour les problèmes simples ou nécessitant un traitement en temps réel. Le pas décroissant offre un bon compromis dans les situations fortement bruitées où la stabilité de la convergence est essentielle.