


Politehnica University of Bucharest
Faculty of Automatic Control and Computers
Department of Automatic Control and Systems Engineering

PHD THESIS

Abstract - Adaptări ale Feasibility Pump pentru Reprezentări Rare

Absolvent
Miertoiu Florin Ilarion

Advisor
Prof. dr. ing. Bogdan Dumitrescu

Bucharest, 2022

1 Introductere

Reprezentarea rară a unui semnal $y \in \mathbb{R}^q$, folosind un dicționar $D \in \mathbb{R}^{m \times n}$, constă în găsirea unei soluţii $x \in \mathbb{R}^n$ având cel mai mic număr posibil de coeficienți nenuli pentru sistemul de ecuatii $y = Dx$. Fiecare coloană a dicționarului D este considerată un atom care poate lua parte la reprezentarea rară x a semnalului y. Dacă valoarea corespunzătoare a lui x pentru un atom al dicționarului D nu este nulă, atunci atomul respectiv face parte din suportul reprezentării rare x a semnalului y. În acest caz, sistemul este subdeterminat cu $m \leq n$. De asemenea, numărul de atomi utilizați în reprezentare este mult mai mic decât numărul de coloane n.

Problema de optimizarea care este luată în considerare în acest caz este

$$\begin{aligned} \underset{x \in \mathbb{R}^n}{\text{minimize}} \quad & \|x\|_0 \\ \text{subject to} \quad & Dx = y \end{aligned} \tag{1.1}$$

Problema (1.1) este NP greu de rezolvat și în practică este modificată pentru ca algoritmul de optimizare liniară să fie utilizat. Constrângerea de egalitate este înlocuită cu măsura nepotrivită $\|y - Hx\|_p \leq u$ unde norma utilizată este convexă ($p \geq 1$), în mai multe cazuri $p = 2$. Deoarece problemele de normă 0 sunt NP-hard, în problemele multiple este înlocuită cu norma 1 care are ca rezultat probleme care pot fi rezolvate în timp polinomial.

Cea mai utilizată adaptare în practică este modelul Least Absolute Shrinkage and Selection Operator (LASSO) [15] care este definit ca

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad \frac{1}{2}\|y - Dx\|_2^2 + \lambda\|x\|_1 \tag{1.2}$$

În acest model, fiecare coloană a matricei D este privită ca o caracteristică, iar vectorul care trebuie reprezentat y este văzut ca rezultatul unui sistem complex. Modelul încearcă să găsească un mic subset de caracteristici care permit o reprezentare precisă a y.

Programele cu numere întregi mixte sunt probleme de optimizare care conțin atât variabile întregi, cât și continue. Dacă toate variabilele sunt numere întregi, problema este o problemă cu numere întregi pură.

Forma generală a programării cu numere întregi mixte este [14]

$$\begin{aligned} \underset{x \in \mathbb{R}^n}{\text{minimize}} \quad & c^T x \\ \text{subject to} \quad & Ax \geq b \\ & x \geq 0 \\ & x_j \in \mathbb{Z}, \forall j \in I \end{aligned} \tag{1.3}$$

cu $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$ și $b \in \mathbb{R}^m$. I este mulțimea care conține indicii variabilelor întregi în x.

Problema originală (1.3) este NP-hard și este mai dificil de rezolvat utilizând un sistem de calcul..

Feasibility Pump, propusă inițial în [9] și [3], este o metodă euristică MIP care generează două secvențe de soluții, una care satisface constrângerile liniare și una care satisface constrângerile întregi. Aceste secvențe sunt generate până când se găsește o soluție care satisface ambele condiții. Feasibility Pump pleacă de la o soluție inițială și apoi continuă, prin mai multe iterații, la minimizarea distanței dintre cele două soluții, mai întâi prin rezolvarea unei probleme de optimizare liniară cu constrângerile întregi eliminate pentru a obține soluția care satisface constrângerea liniară, apoi printr-o etapă de rotunjire pentru a obține soluția care satisface constrângerea întregului. Algoritmul este predispus la ciclări și bucle și mai multe abordări au fost propuse pentru a evita această problemă. [7, 11, 6, 13, 4]. Feasibility Pump oferă timp de execuție mult mai bun (mai multe ordine de mărime față de Branch and Bound sau Branch and Cut), dar calitatea soluției nu este la fel de bună; nu există nicio garanție că optimul este atins.

Luăm în considerare problema MIP

$$\begin{aligned} \underset{x\in\mathbb{R}^n}{\text{minimize}} \quad & c^T x \\ \text{subject to} \quad & Ax \leq b \\ & l \leq x \leq u \\ & x_j \in \mathbb{Z}, \forall j \in I \end{aligned} \tag{1.4}$$

cu A fiind o matrice $m \times n$ și $I \subseteq \{1, 2, ..., n\}$ este setul de index al variabilelor întregi. Pentru a obține o problemă liniară, condiția de integralitate din (1.4) pentru submulțimea $I \subseteq \{1, 2, ..., n\}$ este relaxată la:

$$l_j \leq x_j \leq u_j, \forall j \in I \tag{1.5}$$

Considerăm că o soluție x pentru problema relaxată (1.4) este întreagă, dacă toate elementele x_j sunt întregi pentru toate valorile $j \in I$. Folosind o procedură de rotunjire, se obține un vector întreg $\tilde{x}$ din soluția relaxată x. Definim operatorul care calculează diferența dintre cele două soluții ca

$$\triangle(x, \widetilde{x}) = \sum_{j\in I} |x_j - \widetilde{x}_j| \tag{1.6}$$

Operatorul (1.6) este folosit pentru a forța soluția relaxării problemei (1.4) să fie mai aproape de soluția întreagă $\tilde{x}$. Acest operator poate fi reformulat ca în [1], pentru probleme generale cu numere întregi mixte, ca

$$\begin{aligned} \underset{x\in\mathbb{R}^n}{\text{minimize}} \quad & \triangle(x, \widetilde{x}) = \sum_{j\in I:\widetilde{x}_j=l_j} |x_j - l_j| + \sum_{j\in I:\widetilde{x}_j=u_j} |u_j - x_j| + \sum_{j\in I:l_j\leq\widetilde{x}_j\leq u_j} d_j \\ \text{subject to} \quad & Ax \leq b \\ & d \geq x - \widetilde{x} \\ & d \geq \widetilde{x} - x \\ & l \leq x \leq u \end{aligned} \tag{1.7}$$

unde $d_j = |x_j - \widetilde{x}_j|$ pentru variabilele întregi x_j care nu sunt rotunjite la niciuna dintre cele două limite. Acest termen care nu apare în cazul binar. În acest caz, (1.7) devine [9]

$$\begin{aligned} \underset{x\in\mathbb{R}^n}{\text{minimize}} \quad & \triangle(x,\widetilde{x}) = \sum_{j\in I:\widetilde{x}_j=l_j} |x_j| + \sum_{j\in I:\widetilde{x}_j} |1-x_j| \\ \text{subject to} \quad & Ax \leq b \\ & 0_n^T \leq x \leq 1_n^T \end{aligned} \tag{1.8}$$

Feasibility Pump, care a fost propusă pentru prima dată în [9], folosește ca punct de plecare o soluție fezabilă a problemei relaxate (1.4), din care creează două secvențe de puncte x și $tildex$, unde x este soluția problemei relaxate, dar nu respectă neapărat constrângerea integrală și $\tilde{x}$ care respectă condiția întregului, dar nu este neapărat o soluție a problemei relaxate. Pentru generarea celor două secvențe se aplică următoarea schemă: la fiecare iterație (care în acest algoritm se numește și ciclu de pompare), din soluția relaxată x se obține o nouă soluție întreagă $\tilde{x}$ prin simpla rotunjire a componentei integrale la cel mai apropiat număr întreg sau la o procedură diferită în funcție de problemă, în timp ce o nouă soluție relaxată x se obține prin minimizarea problemei relaxate, definită prin:

$$\begin{aligned} \underset{x\in\mathbb{R}^n}{\text{minimize}} \quad & \triangle(x,\widetilde{x}) \\ \text{subject to} \quad & Ax \leq b \\ & l \leq x \leq u \\ & x_j \in \mathbb{Z}, \forall j \in I \end{aligned} \tag{1.9}$$

Această procedură se aplică până când se găsește o soluție întregă x sau se atinge un anumit număr de iterații. Deoarece această schemă este predispusă la ciclare, este necesar un pas de perturbare [10].

Pașii Feasibility Pump sunt prezentați in Algorithm 1.

Algorithm 1.1: Feasibility Pump

Data: $MIP = min\{c^T x : x \in P, Ax \leq b, l \leq x \leq u, x_j \text{ integer } \forall j \in I\}$
Result: *o soluție fezabilă MIP* $\widetilde{x}$ *(dacă a fost găsită)*

```
1  x = argmin{c^T x : Ax ≤ b, x ∈ P};
2  while nu este satisfăcută condiția de oprire do
3  |  if x este intreg then
4  |  |  return x ;
5  |  else
6  |  |  x̃ = Round(x);
7  |  end
8  |  if se detectează un ciclu then
9  |  |  Perturb(x̃);
10 |  end
11 |  x̄ = argmin{△(x, x̃) : Ax ≤ b, l ≤ x ≤ u, x ∈ P} ;
12 end
```

Algoritmul încearcă să rezolve, la pasul 1, problema inițială relaxată a lui (1.4). Feasibility Pump folosește, la pasul 11, problema relaxată (1.9) pentru fiecare iterație. Soluția problemei relaxate este verificată dacă îndeplinește criteriile de oprire ale algoritmului la pasul 3.

Dacă soluția problemei relaxate nu satisface criteriile de oprire, se utilizează o procedură de rotunjire, pasul 6, pentru a obține soluția întreagă care va fi utilizată pentru următoarea iterație a Feasibility Pump.

Algoritmul este predispus la ciclare. Dacă este detectat un ciclu, soluția întreagă este perturbată la pasul 8. Au fost luate în considerare mai multe strategii de perturbare în funcție de tipul de problemă utilizat.

Deoarece problema inițială nu este prezentă în fiecare ciclu de pompare, calitatea soluției nu este foarte bună deoarece algoritmul este concentrat mai mult pe găsirea soluției întregi. Pentru a rezolva această problemă, Objective Feasibility Pump a fost propusă în [1]. Introduce problema inițială în ciclurile de pompare. Cu fiecare iterație, influența problemei inițiale este redusă pentru a permite algoritmului să converge. Cu această modificare, problema (1.9) este înlocuită cu

$$\begin{aligned} \underset{x\in\mathbb{R}^n}{\text{minimize}} \quad & (1-\alpha)\triangle(x,\widetilde{x}) + \alpha(c^T x) \\ \text{subject to} \quad & Ax \leq b \\ & l \leq x \leq u \\ & x_j \in \mathbb{Z}, \forall j \in I \end{aligned} \tag{1.10}$$

Reducerea influenței problemei inițiale α se face prin înmulțire cu o valoare $\gamma \in (0,1)$ după fiecare etapă a Feasibility Pump:

$$\alpha \leftarrow \gamma\alpha \tag{1.11}$$

Prin adăugarea modificărilor de mai sus, se obține algoritmul Objective Feasibility Pump Algorithm 2.

Algorithm 1.2: Objective Feasibility Pump

Data: $MIP = \min\{c^T x : x \in P, Ax \leq b, l \leq x \leq u, x_j \text{ integer } \forall j \in I\}$
Result: *o soluție fezabilă MIP* $\widetilde{x}$ *(dacă a fost găsită)*

```
x = argmin{c^T x : Ax ≤ b, x ∈ P};
while nu este satisfăcută condiția de oprire do
    if x este intreg then
        return x ;
    else
        x̃ = Round(x);
    end
    if se detectează un ciclu then
        Perturb(x̃);
    end
    x̄ = argmin{(1 − α)△(x, x̃) + α(c^T x) : Ax ≤ b, l ≤ x ≤ u, x ∈ P} ;
    α ← γα ;
end
```

Adăugarea obiectivului în ciclurile de pompare și a parametrului de reducere α necesită o modificare în detecția ciclică a algoritmului. Se poate întâmpla ca, în timpul primelor cicluri de pompare, să se găsească aceeași soluție întreagă. La fiecare pas al algoritmului, soluția întreagă $\tilde{x}$ și α sunt stocate în perechi. Se aplică apoi procedura de repornire sau perturbare, numai dacă la un anumit pas t aflăm că $\alpha_t - \alpha_{t-1} \geq \delta$, unde $delta \in [0,1]$ este setată la începutul algoritmului [1].

2 Adaptări Feasibility Pump Adaptations pentru semnale perturbate

2.1 Adaptări Feasibility Pump Adaptation pentru Reprezentări Rare

Problema de reprezentare rară inițială care se dorește rezolvată cu ajutorul Feasibility Pump este

$$\begin{aligned} \underset{x\in\mathbb{R}^n}{\text{minimize}} \quad & \|y - Dx\|_p \\ \text{subject to} \quad & \|x\|_0 \leq K \end{aligned} \tag{2.1}$$

unde pentru măsurarea erorii de reprezentare a datelor $\|y - Dx\|_p$ poate folosi orice normă p.

Forma problemei, care va fi folosită ca punct de plecare pentru adaptările Feasibility Pump, este problema LASSO

$$\underset{x\in\mathbb{R}^n}{\text{minimize}} \quad \|y - Dx\|_p + \lambda\|x\|_1 \tag{2.2}$$

care încearcă să echilibreze cele două cantități care trebuie minimizate, măsura nepotrivită și numărul de atomi din suport.

Pentru ca algoritmul Feasibility Pump să fie utilizat, o variabilă binară Se introduce $b \in \mathbb{Z}^q$, care arată dacă un atom al dicționarului este folosit pentru a reprezenta semnalul y.

$$\begin{aligned} \underset{x\in\mathbb{R}^n, b\in[0,1]^n}{\text{minimize}} \quad & \|y - Dx\|_p + \lambda\|x\|_1 \\ \text{subject to} \quad & 1_n^T b \leq K \\ & -Mb \leq x \leq Mb \end{aligned} \tag{2.3}$$

Pentru a crea modelul pentru pașii Feasibility Pump, problema (2.2) este combinată cu forma generală a Objective Feasibility Pump (1.10). Problema care se rezolvă la fiecare pas Feasibility Pump devine

$$\begin{aligned} \underset{x\in\mathbb{R}^n, b\in\{0,1\}^n}{\text{minimize}} \quad & (1-\alpha)\triangle(b,\widetilde{b}) + \alpha\left[\beta\|y - Dx\|_p + \lambda\|x\|_1\right] \\ \text{subject to} \quad & 1_n^T b \leq K \\ & -Mb \leq x \leq Mb \end{aligned} \tag{2.4}$$

unde α este parametrul de dezintegrare, utilizat în formularea Obiective Feasibility Pump pentru a crește importanța condiției întregi în comparație cu problema inițială. Reducerea lui α se face prin înmulțire cu o valoare $\gamma \in (0,1)$ ca în (1.11).

Termenul β este folosit pentru a contrabalansa valoarea termenului de regularizare $\|x\|_1$, în comparație cu termenul neadaptat.

Se folosește trucul big-M, ca în articolul [5], unde M este un parametru prestabilit care limitează valoarea lui x, ales ca $M = 1,1\|H^T y\|_\infty / \|H\|_2^2$. Limitează amplitudinea atomilor și ajută la obținerea unui număr mare de coeficienți mai mici care sunt aproape de 0.

Forma generală a algoritmului este similară cu algoritmul 2, problema inițială selectată fiind (2.2) și ciclurile de pompare folosind (2.4).

Algorithm 2.1: Feasibility Pump Modificat

Data: Dictionar $D \in \mathbb{R}^{m \times n}$, semnal de reprezentat $y \in \mathbb{R}^m$, numărul de coeficienți diferiti de zero utilizați pentru reprezentare $K \in \mathbb{Z}$, numărul maxim de iterații $Iter$, ponderi α, γ

Result: o solutie fezabilă s$x \in \mathbb{R}^n$

```
Rezolvă (2.3) pentru b ∈ R^n folosind un solver pentru probleme de optimizare. Vectorii x
 și b sunt obținuți.
Utilizați procedura de rotunjire pentru a obține vectorul b̃.
while număr de iterații ≤ Iter do
    Rezolvați problema (2.8). Vectorii x și b sunt obținuți.
    if b este intreg then
        return x;
    end
    Utilizați procedura de rotunjire pentru a obține vectorul b̃.
    if Se detectează ciclu then
        Utilizați strategia de perturbare selectată activată b̃.
    end
    Actualizați valoarea lui α folosind (1.11).
end
```

Pentru procedura de rotunjire, b real este rotunjit pentru a obține un vector binar $\widetilde{b}$, setând valorile K ale lui b corespunzătoare celor mai mari amplitudini K de x la 1 și celelalte la 0.

Perturbația folosită este o perturbație slabă de forma

$$\widetilde{b}^{k+1} \leftarrow \frac{1}{2}\widetilde{b}^{k+1} + \frac{1}{2}v \tag{2.5}$$

care este folosit pe vectorul întreg $\widetilde{b}$ după ce o anumită condiție este îndeplinită.

2.2 Cazul Zgomotului Laplacian

Pentru cazul perturbației Laplaciane, condiția de egalitate $y = Dx$ este relaxată și înlocuită cu minimizarea măsurării neadaptate a datelor $\|y - Dx\|_1$. Numărul de coeficienți diferiti de zero este delimitat de pragul K iar reprezentarea rară se găsește prin rezolvarea problemei

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad \|y - Dx\|_1 + \lambda \|x\|_1 \tag{2.6}$$

Pentru ca algoritmul Feasibility Pump să fie utilizat, introducem o variabilă binară $b \in \mathbb{Z}^n$, care arată dacă un atom al dicționarului este folosit pentru a reprezenta semnalul y. Fiecare coloană a dicționarului D este asociată cu un element al vectorului $b \in \{0;1\}^n$. Un element din b care are valoarea 0 arată că atomul respectiv al dicționarului nu participă la reprezentare, în timp ce un element cu valoarea 1 arată că acel atom este folosit pentru reprezentarea semnalului y.

Problema (2.6) devine

$$\begin{aligned} \underset{x\in\mathbb{R}^n, b\in\{0,1\}^n}{\text{minimize}} \quad & \|y - Dx\|_1 + \lambda\|x\|_1 \\ \text{subject to} \quad & 1_n^T b \leq K \\ & -Mb \leq x \leq Mb \end{aligned} \tag{2.7}$$

Pentru a crea modelul pașilor de pompare, problema (2.7) este combinată cu forma generală a problemei Objective Feasibility Pump (1.10). Problema care se rezolvă la fiecare pas de pompare devine

$$\begin{aligned} \underset{x\in\mathbb{R}^n, b\in\{0,1\}^n}{\text{minimize}} \quad & (1-\alpha)\triangle(b,\widetilde{b}) + \alpha\left[\beta\|y - Dx\|_p + \lambda\|x\|_1\right] \\ \text{subject to} \quad & 1_n^T b \leq K \\ & -Mb \leq x \leq Mb \end{aligned} \tag{2.8}$$

Această versiune a Feasibility Pump este implementată în MATLAB utilizând *linprog* și comparată cu modelul regularizat de cea mai mică abatere absolută (RLAD) [17] și algoritmul Branch and Cut. Pentru modelul 2.7 sunt luate în considerare două variante, cea care folosește termenul de regularizare și cea fără acesta. Este dovedit că termenul de regularizare ajută algoritmul să aibă rezultate mai bune. De asemenea, este luată în considerare pornirea timpurie pentru a demonstra că poate îmbunătăți timpul de rulare cu o reducere minimă a performanței.

Pentru testele noastre, folosim dicționare generate aleatoriu cu numerele de condiționare 10, 100, 1000, 10000 și 100000. Pentru fiecare număr de condiționare, sunt generate 160 de dicționare de două dimensiuni diferite, 80×200 și 50×100. Semnalele de test se obțin cu $y = Dx_{\text{true}} + u$, unde soluțiile x_{true} au $K \in \{5, 7, 9, 11\}$ coeficienți nenuli generați urmând aleatoriu o distribuție Gaussiană, în poziții aleatorii; zgomotul u este Laplacian si varianta lui se alege astfel incat raporturile semnal/zgomot au valori $10, 20, 30, \infty$.

Pentru calculul erorii de reprezentare, eroarea relativă

$$e = \frac{\|Dx - y\|_1}{\|y\|_1} \tag{2.9}$$

este utilizat (unde acum x este soluția calculată).

Erorile relative de recuperare sunt calculate folosind

$$e_{rec} = \frac{\|x - x_{true}\|_2}{\|y\|_2} \tag{2.10}$$

unde x_{true} este reprezentarea reală a semnalului y folosind dicționarul D pentru cazul neperturbat.

Am implementat SFP și SFPreg așa cum este prezentat în algoritmul 2.1. De asemenea, luăm în considerare o valoare inițială a parametrului $\alpha = 1$. La fiecare iterație α este înmulțit cu un factor de actualizare $\gamma = 0,95$. Numărul de iterații $Iter$ este setat la 1000. Rulăm SFPreg cu 50 de valori ale parametrului de regularizare λ de la 0 la 1. Valoarea pentru care eroarea este cea mai mică este considerată cea mai bună valoare.

Cele două variante ale algoritmului pompei de fezabilitate sunt comparate cu modelul RLAD (deviația absolută minimă regularizată) [17]

$$\underset{x\in\mathbb{R}^n}{\text{minimize}} \quad \|Dx - y\|_1 + \widetilde{\lambda}|x\|_1 \tag{2.11}$$

rezolvat cu *linprog*. Aceasta este o problemă relaxată în stil lasso, adaptată normei de eroare ℓ_1.

Algoritmii sunt implementați în MATLAB și testați pe un computer cu un procesor cu 6 nuclee de 3,4 GHz și 32 GB RAM.

Mai întâi comparăm diferența de eroare, calculată folosind (2.9) între cele două implementări prezentate, SFP și SFPreg. În figurile 2.1 se poate observa că în majoritatea cazurilor SFPreg are o eroare mai mică decât SFP. Este evident că SFPreg este un algoritm mai bun având o reprezentare cu o eroare mai mică decât SFP. Atât SFP, cât și SFPreg se recuperează fără eroare în cazul în care $SNR = \infty$.

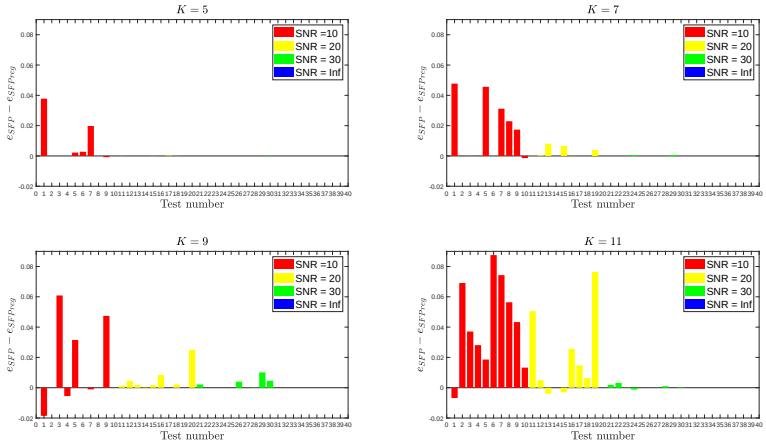

Figure 2.1: Mean representation error difference between SFP and SFPreg for a dictionary conditioning of 10

În figurile 2.2 este afișată diferența dintre erorile medii de recuperare calculate folosind (2.10). Comportamentul este similar ca și în cazul erorii de reprezentare, SFPreg fiind mai bun decât SFP în majoritatea cazurilor.

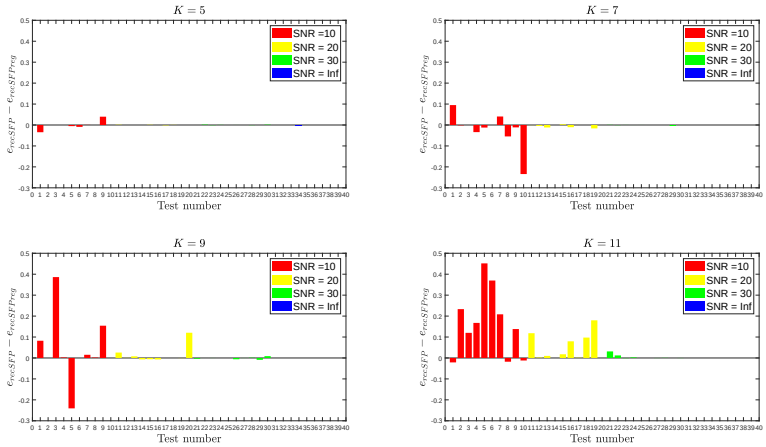

Figure 2.2: Mean recovery error difference between SFP and SFPreg for a dictionary conditioning of 10

Se poate observa că diferențele erorilor de reprezentare sunt mult mai mici decât diferența erorilor de recuperare, ceea ce indică faptul că SFP are mai multi coeficienti fals pozitive în suportul soluției decât SFPreg. În cazul neperturbat, ambii algoritmi recuperează suportul adevărat.În cazurile perturbate, SFPreg are 1,32 coeficienti fals pozitivi pe caz, în timp ce SFP are 1 coeficient fals pozitiv pe caz. SFP recuperează suportul real în 294 de cazuri din

600 de cazuri (49%), în timp ce SFPreg recuperează suportul real în 317 din 600 de cazuri (53%). Un coeficient fals negativ indică când un atom lipsește din suportul soluției adevărate. Un coeficient fals pozitiv apare atunci când sprijinul soluția calculată conține atomi în afara suportului adevărat.

SFP necesită 0,5 secunde pentru a rula cazul neperturbat și 3,6 secunde pentru cazul perturbat, în timp ce SFPreg necesită 0,6 secunde în cazul neperturbat și 1,8 secunde pentru cazul perturbat. Adăugarea parametrului de regularizare îmbunătățește timpul de rulare al algoritmului. Acest lucru poate fi observat și pentru numărul de iterații utilizate pentru cazurile perturbate, SFPreg utilizând 5,4 iterații și SFP având nevoie în medie de 16,2 iterații.

RLAD găsește suportul adevărat în toate cazurile neperturbate, dar în cazul perturbat găsește suportul adevărat în doar 46 din cele 600 (7,7%) cazuri perturbate. RLAD indică 3,03 coeficienti fals negativi per caz pertubat. De asemenea, RLAD prezintă coeficienti fals pozitivi în 543 din 600 (90,5%) cazuri perturbate, cu o medie de 0,9 coeficienti fals pozitivi per caz. RLAD recuperează suportul la fel de bine ca SFP și SFPreg în cazul neperturbat, dar are rezultate sub ambii algoritmi în cazurile perturbate.

În medie, implementarea Branch and Bound necesită 165 de secunde pentru a rula un caz neperturbat și 360 de secunde pentru cazul perturbat. Recuperează suportul adevărat în toate cazurile neperturbate și în 336 din 600 de cazuri perturbate (56%). Branch and Bound indică în medie 0,87 coeficienti fals negativi per caz, în cazurile perturbate. Branch and Bound oferă rezultate ușor mai bune în comparație cu SFPreg, dar timpul de rulare este mai mare cu trei ordine de mărime.

Table 2.1: Eroare medie pentru algoritmi pentru numărul de condiționare 10

K	SNR	SFP $\alpha = 0.5$	SFPreg $\alpha = 0.5$	SFP $\alpha = 1$	SFPreg $\alpha = 1$	RLAD	BB
$K = 5$	$SNR = 10$	0.292	0.266	0.266	**0.260**	0.497	0.262
	$SNR = 20$	0.078	0.078	0.078	0.078	0.260	**0.077**
	$SNR = 30$	**0.026**	**0.026**	**0.026**	**0.026**	0.160	**0.026**
	$SNR = \infty$	$9.341 \cdot 10^{-10}$	$5.632 \cdot 10^{-13}$	$3.252 \cdot 10^{-10}$	$4.474 \cdot 10^{-13}$	$1.720 \cdot 10^{-10}$	$\mathbf{4.263 \cdot 10^{-16}}$
K=7	$SNR = 10$	0.306	0.249	0.245	**0.229**	0.413	0.233
	$SNR = 20$	0.089	0.082	0.083	0.081	0.341	**0.081**
	$SNR = 30$	0.027	**0.026**	**0.026**	**0.026**	0.283	**0.026**
	$SNR = \infty$	0.019	$1.807 \cdot 10^{-13}$	$1.218 \cdot 10^{-10}$	$7.512e \cdot 10^{-14}$	$3.003 \cdot 10^{-10}$	$\mathbf{6.399 \cdot 10^{-16}}$
K=9	$SNR = 10$	0.329	0.228	0.217	0.205	0.518	**0.207**
	$SNR = 20$	0.109	0.078	0.080	0.076	0.433	**0.075**
	$SNR = 30$	0.063	**0.026**	0.028	0.026	0.332	**0.026**
	$SNR = \infty$	0.037	$7.530 \cdot 10^{-13}$	$1.390 \cdot 10^{-10}$	$5.701 \cdot 10^{-13}$	$4.293 \cdot 10^{-10}$	$\mathbf{5.916 \cdot 10^{-16}}$
K=11	$SNR = 10$	0.317	0.252	0.227	**0.184**	0.454	0.189
	$SNR = 20$	0.150	0.076	0.089	**0.072**	0.335	**0.072**
	$SNR = 30$	0.082	**0.024**	0.025	0.025	0.316	**0.024**
	$SNR = \infty$	0.086	$6.575 \cdot 10^{-13}$	$3.116 \cdot 10^{-10}$	$2.599 \cdot 10^{-13}$	$3.829 \cdot 10^{-10}$	$\mathbf{7.676 \cdot 10^{-16}}$

Algoritmul SFPreg oferă o recuperare mai bună a erorilor decât RLAD și SFP. De asemenea, pare să ofere un compromis bun, în comparație cu implementarea Branch și Bound, deoarece oferă o recuperare a suportului la fel de bună, dar cu un timp de execuție mult mai bun.

2.3 Cazul Zgomotului Gaussian

Pentru a obține reprezentarea rară x a unui semnal y afectat de o perturbație Gaussiană, în ecuația (2.1) este utilizată măsura nepotrivită a datelor $\|y - Dx\|_2$. Numărul de coeficienți

diferiti de zero este mărginit în mod similar de pragul K și reprezentarea rară se găsește prin rezolvarea problemei

$$\underset{x\in\mathbb{R}^n}{\text{minimize}} \quad \|y - Dx\|_2^2 + \lambda\|x\|_1 \tag{2.12}$$

Ne propunem să combinăm abordarea MIP cu problema lasso (2.12). Variabila binară $b \in \{0,1\}^n$ este introdus pentru a îndeplini rolul unui indicator care arată ce atom al dicționarului D este folosit pentru reprezentarea lui y. Apoi combinăm (**??**) cu (2.12) pentru a obține problema

$$\begin{aligned} \underset{x\in\mathbb{R}^n, b\in\{0,1\}^n}{\text{minimize}} \quad & \|y - Dx\|_2^2 + \lambda\|x\|_1 \\ \text{subject to} \quad & 1_n^T b \le K \\ & -Mb \le x \le Mb \end{aligned} \tag{2.13}$$

care va fi formularea problemei care va fi analizată în acest capitol și va fi utilizată pentru modificările algoritmului Feasibility Pump.

Pentru a implementa problema (2.13), pentru termenul de regularizare se introduce o variabilă auxiliară $w \in \mathbb{R}^n$, Rezultând

$$\begin{aligned} \underset{x\in\mathbb{R}^n, w\in\mathbb{R}^n, b\in\{0,1\}^n}{\text{minimize}} \quad & x^T D^T Dx - 2y^T Dx + \lambda 1_n^T w \\ \text{subject to} \quad & 1_n^T b \le K \\ & -w \le x \le w \\ & w \le Mb \end{aligned} \tag{2.14}$$

În fiecare iterație a algoritmului Feasibility Pump, la pasul 4, vectorul b și soluția actuală x sunt actualizate prin rezolvare

$$\begin{aligned} \underset{x\in\mathbb{R}^n, w\in\mathbb{R}^n, b\in[0,1]^n}{\text{minimize}} \quad & (1-\alpha)\triangle(b,\widetilde{b}) + \alpha\left[\frac{K}{err_{init}^2}(x^T D^T Dx - 2y^T Dx) + \lambda 1_n^T w\right] \\ \text{subject to} \quad & 1_n^T b \le K \\ & -w \le x \le w \\ & w \le Mb \end{aligned} \tag{2.15}$$

unde $\triangle(b,\widetilde{b}) = \|b - \widetilde{b}\|_1$ și err_{init} este eroarea de reprezentare la pasul 1 al algoritmului **??**. Această problemă de programare pătratică este rezolvată și cu quadprog în MATLAB.

În timpul testelor noastre, s-a observat că adăugarea termenului $\frac{K}{err_{iter}^2}$ în (2.15) este necesară pentru a crește influența erorii în procesul de optimizare pentru o perioadă mai lungă de timp în timpul testelor, deoarece este mult mai mic decât termenii $\triangle(b,\widetilde{b})$ și $\|x\|_1$ și are nevoie de un parametru suplimentar de greutate.

Această versiune a Feasibility Pump este comparată cu diverși algoritmi proiectați pentru semnale afectate de zgomotul gaussian. Algoritmii utilizați pentru comparare sunt OMP (Orthogonal Matching Pursuit) [16], FISTA (Fast Iterative Shrinkage-Thresholding Algorithm) și modificarea ADMM pentru Feasibility Pump prezentată în [11]. Se arată că Feasibility Pump oferă rezultate mai bune în majoritatea cazurilor în comparație cu ceilalți algoritmi.

Pentru calcularea erorii de reprezentare se folosește o eroare relativă, care folosește norma l_2

$$e = \frac{\|Dx - y\|_2}{\|y\|_2} \tag{2.16}$$

este utilizat (unde acum x este soluția calculată).

Am implementat versiunea pentru carcasa de zgomot Gaussian, denumită SQFP.. Greutatea inițială este setată ca $\alpha = 1$ și este înmulțită cu un factor de actualizare $\gamma = 0,9$ la fiecare iterație; aceste valori par să ofere un compromis bun între viteza de convergenţă şi eroarea de reprezentare. Folosim $\gamma = 0.9$ în acest capitol, nu $\gamma = 0.95$, deoarece modelul este foarte lent cu adăugarea de $\frac{K}{err^2_{iter}}$ și permite viteza simularea noastră să fie mai bună, pierderea de precizie fiind mică. Numărul de iterații $Iter$ este setat la 1000. Rulăm SQFP cu 50 de valori egal distanțate ale parametrului de regularizare λ de la 0 la 1. Valoarea pentru care eroarea medie de reprezentare este cea mai mică și este considerată cea mai bună alegere pentru λ. Dicționarele au fost generate la fel ca în cazul Laplacian.

Algoritmii care sunt utilizați pentru comparație sunt OMP (Orthogonal Matching Pursuit) [16], FISTA (Algoritmul de contracție rapidă iterativă) [2] și FP-ADM [11].

Erorile medii, obținute prin rularea testelor, sunt afișate în cifre 2.3. Prima bară (roșie) înăuntru fiecare celulă corespunde erorii relative a FISTA, a doua (verde) este pentru SQFP, a treia (albastru) este pentru OMP și ultimul (galben) este FP-ADM. Se poate observa că, pe măsură ce nivelul de sparsitate K crește, algoritmul SQFP are o valoare mult mai mică. eroare de reprezentare decât FISTA și, numai pentru unele numere de condiție, decât OMP și FP-ADM. Pentru $K = 5$ și $K = 7$, diferența dintre algoritmi este foarte mică. Pentru K mai mari, SQFP arată că este în mod clar algoritmul mai bun.

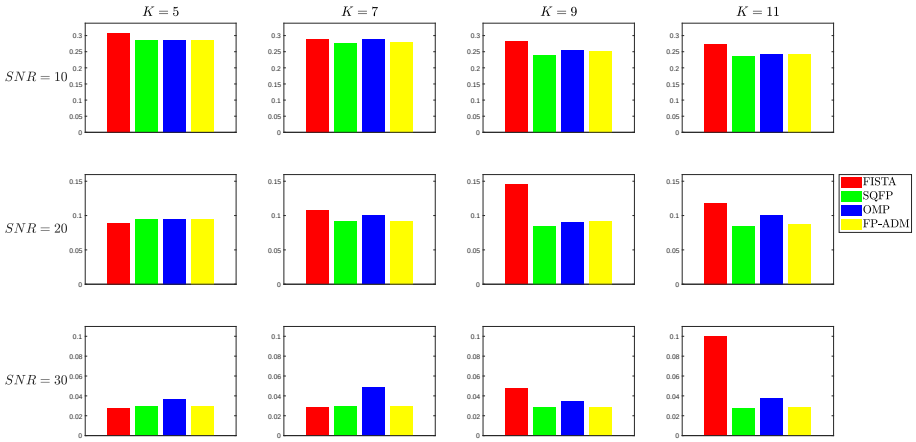

Figure 2.3: Mean Errors for the algorithms tested for a dictionary conditioning of 10

Eroriile medii de recuperare al celor patru algoritmi sunt prezentate în figurile 2.4. Similar cu cazul eroarilor medii, SQFP are o eroare de recuperare mult mai bună decât algoritmul FISTA și, în mai multe cazuri, mai bună decât OMP și FP-ADM. Încă o dată FP-ADM urmează comportamentul SQFP, dar cu o eroare mai mare.

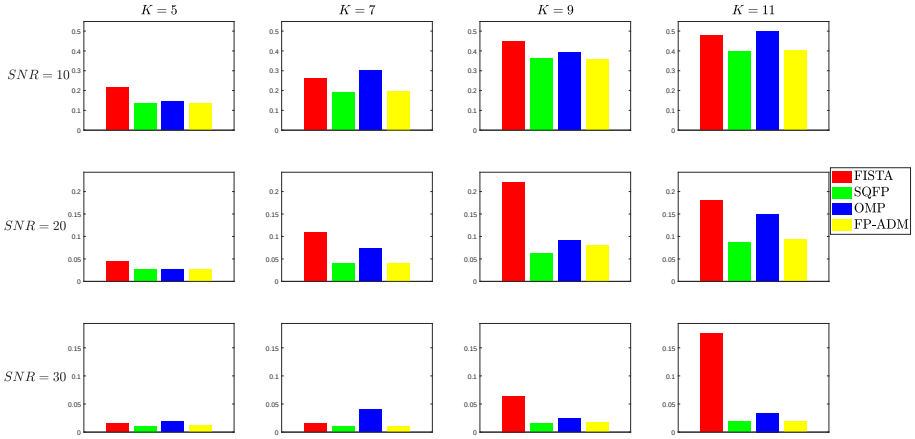

Figure 2.4: Mean Recovery Errors for the algorithms tested for a dictionary conditioning of 10

În tabelele 2.2 sunt afișate erorile medii ale algoritmilor pentru fiecare K și SNR, pentru numărul de condiționare 10. Se poate observa că în cazul semnalului de reprezentare neperturbată doar SQFP reușește să recupereze semnal în toate cazurile. Se poate observa că toți ceilalți algoritmi au cazuri în care nu recuperează semnalul neperturbat.

Table 2.2: Eroare medie pentru algoritmii pentru numărul de condiționare 10

K	SNR	SQFP	FISTA	OMP	FP-ADMM
$K = 5$	$SNR = 10$	**0.285**	0.308	0.286	**0.285**
	$SNR = 20$	**0.095**	0.107	0.095	**0.095**
	$SNR = 30$	0.030	**0.029**	0.036	0.030
	$SNR = \infty$	$2.451 \cdot 10^{-16}$	$\mathbf{1.615 \cdot 10^{-16}}$	$2.451 \cdot 10^{-16}$	$2.451 \cdot 10^{-16}$
$K = 7$	$SNR = 10$	**0.277**	0.289	0.288	0.280
	$SNR = 20$	**0.092**	0.107	0.100	**0.092**
	$SNR = 30$	0.030	**0.029**	0.049	0.030
	$SNR = \infty$	$2.815 \cdot 10^{-16}$	$\mathbf{2.318 \cdot 10^{-16}}$	0.022	0.005
$K = 9$	$SNR = 10$	**0.239**	0.282	0.254	0.253
	$SNR = 20$	**0.085**	0.145	0.091	0.092
	$SNR = 30$	**0.028**	0.048	0.035	0.029
	$SNR = \infty$	$\mathbf{2.826 \cdot 10^{-16}}$	0.025	0.008	$4.008 \cdot 10^{-4}$
$K = 11$	$SNR = 10$	**0.235**	0.275	0.242	0.243
	$SNR = 20$	**0.084**	0.119	0.101	0.087
	$SNR = 30$	**0.028**	0.100	0.038	0.029
	$SNR = \infty$	$\mathbf{3.315 \cdot 10^{-16}}$	0.025	0.053	0.013

În medie, numărul de iterații necesare algoritmului SQFP este de aproximativ 4,29 pentru cazurile neperturbate și 22,52 pentru cazurile perturbate. De asemenea, pasul de perturbare apare doar în 6 din cele 600 de cazuri. Suportul adevărat este recuperat pentru toate cazurile neperturbate, iar în cazul perturbat numărul mediu de atomi, care lipsesc din suport, este de 1,32, suportul fiind recuperat corect în 305 din cele 600 de cazuri perturbate (51%). Timpul mediu de rulare este de 0,64 secunde pentru cazul neperturbat și 2,4 secunde pentru cazul perturbat. De asemenea, SQFP găsește, în medie, 0,94 coeficenti fals negativi în cazul perturbat.

FISTA necesită, în medie, 1352 de iterații pentru cazul neperturbat și 558 în cazul perturbat. Timpii medii de rulare au fost de 0,03 secunde pentru cazul neperturbat și 0,13 secunde pentru cazul perturbat. În cazul neperturbat, suportul este recuperat în 158 cazuri din 200

(79%) și 161 din 600 în cazul perturbat (26,8%). FISTA arată un coeficient fals pozitiv în 160 (80%) cazuri neperturbate și în 534 (89%) cazuri perturbate. FISTA găsește, în medie, 0,56 coeficienți false negativi în cazul neperturbat și 2,38 fals negative în cazul perturbat. FISTA găsește, de asemenea, coeficienți fals pozitive în 64 din 200 (64%) cazuri neperturbate și în 119 din 600 cazuri perturbate (19,8%), având în medie 1 coeficient fals pozitiv pe caz atât pentru cazurile perturbate, cât și pentru cele neperturbate.

Algoritmul OMP utilizează un număr mediu de iterații care este egal cu numărul de atomi din dicționarul utilizat pentru reprezentare. În cazul neperturbat, OMP găsește suportul corect în 149 de cazuri din 200 (74,5% din cazuri) și în 226 cazuri din 600 pentru cazurile perturbate (37,6% din cazuri). Atât pentru cazul perturbat, cât și pentru cel neperturbat, timpul de rulare este, în medie, în jur de 0,001 secunde. În medie, OMP are 0,67 coeficienți fals negativi în cazurile neperturbate și 1,68 coeficienți fals negativi în cazurile perturbate.

Algoritmul FP-ADM folosește 1000 de iterații în fiecare caz pentru a obține reprezentările (algoritmul se oprește doar la limita numărului de iterație). Timpul mediu de rulare este de 0,58 secunde pentru cazul neperturbat și 0,49 pentru cazul perturbat. Algoritmul recuperează suportul adevărat în 164 din cele 200 de cazuri (82% din cazuri) pentru cazul neperturbat și în 268 din 600 dintre cazurile perturbate (44,7%). FP-ADM are cazuri în care nu recuperează semnalul în cazul neperturbat. În medie, FP-ADM are 0,21 coeficienți falsi negativi în cazul neperturbat și 1,06 coeficienți falsi negativi în cazul perturbat.

Se poate observa că ambii algoritmi bazați pe FP au rezultate mai bune decât OMP și FISTA. SQFP se dovedește a fi mai bun decât FP-ADM având o eroare medie și o eroare medie de recuperare mai bune.

2.4 Cazul Zgomotului Gaussian Generalizat

Problemele cu zgomotul Gaussian generalizat conduc, prin probabilitate maximă, la optimizare care implică norma p. Acest lucru este de obicei convenabil pentru $p \geq 1$. Totuși, pentru $p < 1$ problemele devin neconvexe și mai greu de rezolvat.

Problema de reprezentare rară asociată cu GG(p) este ecuația (2.17)

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad \|y - Dx\|_p + \lambda \|x\|_1 \tag{2.17}$$

În cele mai multe cazuri, se presupune că parametrul de formă p este cunoscut. Cu toate acestea, utilizarea unui p diferit poate duce la erori de aproximare mai mari. Doar dacă valoarea parametrului de formă este aproape de cea adevărată, putem spera să obținem o soluție x care este mai aproape de soluția adevărată.

Propunem algoritmul 2.2, care constă într-o modificare pentru Feasibility Pump (FP).

Algoritmul rezolvă în mod repetat două probleme, în care variabila binară b este relaxată la intervalul $[0,1]^n$. Soluția este apoi rotunjită la $\widetilde{b}$, cel mai apropiat vector binar cu K valori de unu. Una dintre probleme este (2.17), care este reformulat ca un program de optimizare convex

$$\begin{aligned} \underset{x \in \mathbb{R}^n, b \in [0,1]^n}{\text{minimize}} \quad & \|y - Dx\|_p + \lambda \|x\|_1 \\ \text{subject to} \quad & 1_n^T b \leq K \\ & -Mb \leq x \leq Mb \end{aligned} \tag{2.18}$$

Algorithm 2.2: Feasibility Pump Modificat

Data: Semnal de reprezentat $y \in \mathbb{R}^m$, dicționar $D \in \mathbb{R}^{m \times n}$, nivel de sparsitate $K \in \mathbb{Z}$, număr maxim de iterații $Iter$, ponderi α, λ, γ

Result: Reprezentare rară $x \in \mathbb{R}^n$

1 Rezolvați problema relaxată (2.18) cu $b \in [0,1]^n$. Vectorii x și b sunt obținuți.
2 Utilizați procedura de rotunjire pentru a obține vectorul $\widetilde{b}$.
3 **while** *număr de iterații* $\leq$ *Iter* **do**
4 Rezolvați problema ((2.19)) pentru x și b.
5 **if** *b este întreg* **then**
6 exit loop;
7 **end**
8 Utilizați procedura de rotunjire pentru a obține vectorul$\widetilde{b}$
9 **if** *ciclu se detectează* **then**
10 Perturbă $\widetilde{b}$
11 **end**
12 Actualizați $\alpha \leftarrow \gamma\alpha$
13 **end**
14 Returnează x, optimizat cu un Program de optimizare liniară cu norma p pe suportul găsit.

care reprezintă punctul de plecare al Feasibility Pump și se rezolvă la pasul 1.

Etapele Feasibility Pump folosesc modelul

$$\begin{aligned} \underset{x\in\mathbb{R}^n, b\in[0,1]^n}{\text{minimize}} \quad & (1-\alpha)\triangle(b,\widetilde{b}) + \alpha\left[\|y - Dx\|_p + \lambda\|x\|_1\right] \\ \text{subject to} \quad & 1_n^T b \leq K \\ & -Mb \leq x \leq Mb \end{aligned} \tag{2.19}$$

care se rezolvă la pasul 4 al algoritmului 2.2.

Modelele (2.18) și (2.19) sunt implementate folosind CVX [12] și sunt comparate cu algoritmul greedy OMP-ℓ_p si algoritmul prezentat în [18], care se numește LP_L1.

Rezultatele numerice sunt obținute folosind o schemă de testare similară cu cea utilizată în capitolele precedente, cu distincția semnificativă că zgomotul de perturbare este acum Gaussian Generalizat.

Algoritmul Feasibility Pump adaptat pentru zgomotul Gaussian Generalizat, numit FP-GGN, este comparat cu modificarea OMP prezentată în [19] numită OMP-p și cu algoritmul prezentat în [18], care se numește LP_L1.

Dicționarele și semnalele de testare sunt generate în mod similar ca în cazul Laplacian. Valoarea λ selectată pentru FP-GGN este fixată la 0,008, iar pentru LP_L1 este fixată la 0,0025.

Pentru a calcula eroarea de reprezentare se folosește o formulă care utilizează norma l_p

$$e = \frac{\|Dx - y\|_p}{\|y\|_p} \tag{2.20}$$

unde x este soluția calculată.

Erorile relative de recuperare sunt calculate folosind

$$e_{rec} = \frac{\|x - x_{true}\|_p}{\|y\|_p} \tag{2.21}$$

care este o formulă similară cu (2.10), dar norma l_p înlocuiește l_2, unde x_{true} este reprezentarea adevărată a semnalului y folosind dicționarul D pentru cazul neperturbat.

Figurile 2.5 arată eroarea medie de reprezentare a FP-GGN, calculată folosind (**??**) pentru toate cazurile de testare cu $K = 5$ și condiția numărul 10. Se poate observa că pentru toate cazurile, erorile urmăresc amplitudinea zgomotului pentru toate valorile p. Comportamentul este similar cu modelele utilizate pentru cazul de zgomot Laplacian și pentru cazul de zgomot Gaussian.

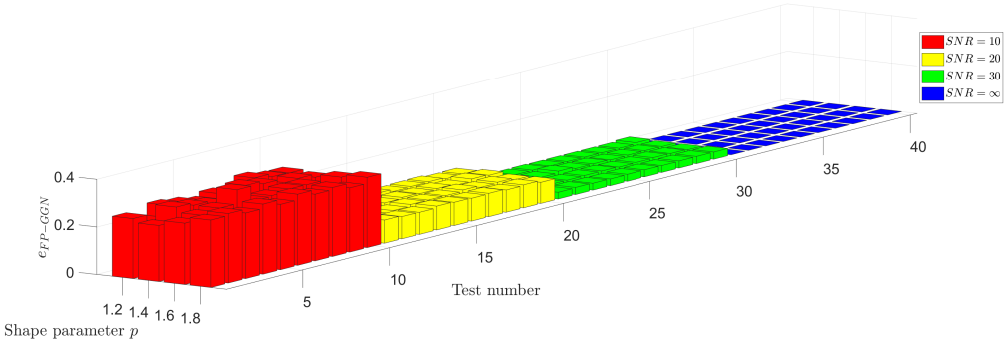

Figure 2.5: Mean representation errors for all the tested norms for FP-GGN for $K = 5$ and condition number 10

Erorile medii de reprezentare sunt afișate în tabelul 2.3, pentru condiția numere 10 și pentru $K = 5$. Se poate observa că doar FP-GGN recuperează semnalul neperturbat, în timp ce LP_L1 pare să aibă cea mai mare dificultate în acest caz. În cazul perturbat, OMP-p are cele mai mari erori de recuperare în majoritatea cazurilor, în special pentru SNR-urile mici. În majoritatea cazurilor, FP-GGN oferă cea mai bună eroare medie de recuperare, dar LP_L1 are cazuri în care oferă o eroare medie de recuperare mai bună, în special pentru SNR-urile mari.

Table 2.3: Eroare medie pentru OMP-p, FP-GGN și LP_L1 pentru numărul de condiționare 10 și $K = 5$

p	SNR	FP-GGN	OMP-p	LP_L1
$p = 1.2$	$SNR = 10$	0.262	0.347	**0.261**
	$SNR = 20$	0.089	0.099	**0.088**
	$SNR = 30$	0.028	0.028	**0.027**
	$SNR = \infty$	$\mathbf{8.596 \cdot 10^{-10}}$	0.001	0.030
$p = 1.4$	$SNR = 10$	0.277	0.341	**0.267**
	$SNR = 20$	**0.092**	0.100	**0.092**
	$SNR = 30$	0.030	0.030	**0.029**
	$SNR = \infty$	$1.167 \cdot 10^{-9}$	$7.169 \cdot 10^{-4}$	$\mathbf{2.708 \cdot 10^{-16}}$
$p = 1.6$	$SNR = 10$	0.283	0.359	**0.276**
	$SNR = 20$	**0.091**	0.094	**0.091**
	$SNR = 30$	**0.030**	0.031	**0.030**
	$SNR = \infty$	$\mathbf{8.624 \cdot 10^{-10}}$	0.001	0.039
$p = 1.8$	$SNR = 10$	0.286	0.337	**0.278**
	$SNR = 20$	0.095	0.099	**0.094**
	$SNR = 30$	**0.029**	0.029	**0.029**
	$SNR = \infty$	$2.088 \cdot 10^{-9}$	0.002	$\mathbf{2.841 \cdot 10^{-16}}$

În cazurile în care $p = 1,8$, FP-GGN are nevoie, în medie, de 3,9 secunde (3 iterații)

pentru cazul neperturbat și 5,3 secunde (3,7 iterații) pentru cazul perturbat. LP_L1 are o durată medie de rulare de 0,44 secunde (53 iterații) pentru cazul neperturbat și 0,70 secunde (224 iterații) pentru cazul perturbat. OMP-p folosește 0,51 secunde pe caz neperturbat și 0,74 secunde pe caz perturbat.

FP-GGN recuperează suportul adevărat în toate cazurile neperturbate și în 265 din cele 600 de cazuri perturbate (44,1%). LP_L1 recuperează suportul adevărat în 170 din 200 de cazuri neperturbate (85%) și 204 din 600 de cazuri perturbate (34%). OMP-p recuperează suportul în 164 din 200 de cazuri neperturbate (82%) și în 255 din 600 (42,5%) cazuri perturbate.

FP-GGN are, în medie, 1,14 coeficienți fals negativi per caz perturbat, în timp ce LP_L1 are 0,16 coeficienți fals negativi pe caz neperturbat și 1,67 coeficienți fals negativi per caz neperturbat. OMP-p are 0,5 coeficienți fals negativi per caz neperturbat și 1,30 coeficienți fals negativi per caz perturbat.

În majoritatea testelor, FP-GGN are cea mai mică recuperare medie dintre toți cei trei algoritmi și, de asemenea, recuperează semnalul mult mai bine decât OMP-p și LP_L1, având cel mai mic număr de fals negative.

3 Estimarea parametrului de formă

3.1 Cadrul general de estimare a parametrilor de formă

În general, un algoritm de reprezentare rară (ARR) are o presupunere de bază a caracteristicilor zgomotului.

Presupunând că avem zgomot doar în cadrul familiei GG, cu parametrul de formă necunoscut $p_{\text{true}} \geq 1$, există și ARR-uri care pot funcționa cu orice p dat, cum ar fi OMP-p și LP_L1. Problema este că parametrul de formă p_{true} nu este adesea cunoscut a priori.

Deci, în majoritatea cazurilor, dacă calculăm eroarea $y - Dx$ obținută de un FSRA și apoi aplicăm algoritmul de estimare a parametrilor de formă (SPE) din [8], parametrul de formă rezultat $\tilde{p}$ este clar diferit de p_{true} precum și de p pe care l-am folosit.

Scopul nostru este estimarea p_{true} folosind un FARR.

Ideea pentru ambele versiuni este să începem cu un p dat (începem de la 2, deoarece în majoritatea cazurilor zgomotul este gaussian). La fiecare pas al algoritmului nostru iterativ, eroarea $y - Dx$ este calculată și algoritmul SPE [8] este utilizat pentru această eroare pentru a calcula $\tilde{p}$ asociat.

Diferența dintre cele două versiuni ale algoritmului este dată de actualizările parametrului de formă p. Pentru prima variantă a algoritmului, numită Half-Adapt, norma este actualizată prin

$$p \leftarrow (p + \tilde{p})/2. \tag{3.1}$$

Deci, mergem către p_{true}, ghidați de distribuția empirică a zgomotului, dar temperăm schimbarea în p pentru a preveni oscilațiile. Reprezentarea rară este calculată cu noul p și așa mai departe.

A doua variantă a algoritmului, denumită Mean-Norm, norma este actualizată prin

$$p \leftarrow \tilde{p}. \tag{3.2}$$

Pentru această a doua variantă a algoritmului, la sfârșit se adaugă un pas suplimentar în care p este calculat ca medie a tuturor valorilor lui p care au fost găsite în timpul fiecărei iterații folosind algoritmul de la [8], cu excepția pentru $p = 2$ care este setat la pasul inițial. După ce p este calculat, FARR este rulat încă o dată cu norma medie găsită pentru a obține eroarea.

Algoritmul general utilizat pentru estimarea SPE este reprezentat de 3.1.

Algorithm 3.1: Forma generală pentru algoritmul de estimare a parametrilor de formă

Data: Semnal de reprezentat $y \in \mathbb{R}^{m \times t}$, dictionar $D \in \mathbb{R}^{m \times n}$, nivel de sparsitate $K \in \mathbb{Z}$, număr maxim de iterații pentru estimarea normei ℓ_p $Iter_{norm}$, limită de oprire θ

Result: Reprezentare rară $x \in \mathbb{R}^{n \times t}$, parametrul de formă estimat $p \in \mathbb{R}$, $p \geq 1$

1 Calculați reprezentări rare x folosind un algoritm care folosește norma $p = 2$ pentru fiecare coloană y

2 Folosește algoritmul din [8] pentru a estima parametrul de formă $\tilde{p}$ din erorile de reprezentare

3 Actualizează norma p folosind (3.1) sau (3.2)

4 **while** *număr de iterații* $\leq Iter_{norm}$ *sau* $|p - \tilde{p}| > \theta$ **do**

5 Calculați reprezentări rare x folosind algoritmul cu p de la pasul anterior

6 Utilizați algoritmul în [8] pentru a estima parametrul de formă $\tilde{p}$

7 Actualizați p folosind (3.1), Half-Adapt, sau (3.2), Mean-Norm

8 **end**

9 Dacă se folosește varianta Mean-Norm, numărul de iterații a fost mai mare decât 1, se calculează norma p ca valoare medie a tuturor valorilor anterioare ale lui $\tilde{p}$. Calculați reprezentări rare x folosind algoritmul cu p obținut.

Aceste cadre sunt concepute astfel încât orice FARR să poată fi utilizat cu el. Algoritmul începe prin rularea FARR, la pasul 1, pentru norma l_2. Folosind algoritmul SRE, $\tilde{p}$ este calculat la pasul 2. Noul p este calculat folosind $\tilde{p}$ cu actualizarea (3.1) sau (3.2) la pasul 3. La fiecare iterație, FARR este evaluat, la pasul 4, cu noua valoare de p pentru SPE și ciclul se repetă. Algoritmul se oprește după ce diferența dintre $|p - \tilde{p}|$ este sub un anumit prag θ. Deoarece este nerezonabil să urmărim o estimare foarte precisă a p_{true}, păstrăm θ suficient de mare. De asemenea, limităm numărul de pași la $Iter_{norm}$, pentru a opri eventualele comportamente neregulate. După oprirea algoritmului, la pasul 9, dacă se utilizează actualizarea (3.2), după finalizarea buclei principale, se calculează media tuturor valorilor obținute de p și se evaluează FARR pentru valoarea obținută. Aceasta este valoarea finală pentru p.

Ambele adaptări sunt testate pentru Generalized Gaussian Feasibility Pump, algoritmul OMP-ℓ_p [19] și algoritmul LP_L1 [18]. De asemenea, sunt comparați cu versiunile de algoritmi care cunosc adevărata valoare p. Se arată că ambele versiuni pot găsi parametri de formă care sunt aproape de valoarea p_{true}. De asemenea, reprezentarea medie și eroarea medie de recuperare sunt similare cu algoritmii care folosesc valoarea reală a p. De asemenea, se poate observa că Mean-Norm este mai rapid decât Half-Adapt și oferă, de asemenea, o mai bună recuperare a suportului.

Configurația de testare este similară cu cea utilizată pentru cazul zgomotului Gaussian generalizat, cu dicționare de dimensiune 50×100 și folosind aceleași valori pentru K, SNR, λ și p. Pentru fiecare combinație K, p, testăm 5 seturi de date diferite. Algoritmii care sunt integrați în cele două variante vor fi FP-GGN, OMP-p și LP_L1.

Algoritmii sunt comparați în ceea ce privește erorile medii de reprezentare, erorile de recuperare și parametrii de formă estimați. Eroarea relativă de reprezentare este calculată folosind (2.20), iar eroarea de recuperare este calculată folosind (2.21).

Pentru primul caz, SNR-ul pentru semnalul de perturbație este setat la 30. Estimarea parametrului de formă este analizată pentru 5 rulări. Pentru toți cei trei algoritmi sunt testați cele două cadre. În tabelul 3.1, valoarea normei, pentru fiecare pereche de celule Half-Adapt și Mean-Norm, pentru valoarea lor respectivă de K și p, care este cea mai apropiată de valoarea reală a p, este scris cu caractere aldine. Se poate observa că Mean-Norm oferă cea mai apropiată aproximare a valorii reale a p în majoritatea cazurilor. Pentru $K = 5$ și $K = 7$, FP-GGN oferă cea mai apropiată aproximare. Pe măsură ce valorile cresc, LP_L1 pare să aibă cea mai apropiată aproximare.

Table 3.1: Parametrii de formă estimați medii pentru OMP-p (sus în fiecare celulă), FP-GGN (mijloc) și LP_L1 (jos)

K	5		7		9		11	
	Half-Adapt	Mean-Norm	Half-Adapt	Mean-Norm	Half-Adapt	Mean-Norm	Half-Adapt	Mean-Norm
p=1	1.273	1.171	1.229	1.232	1.230	1.218	**1.141**	1.267
	1.139	1.133	1.210	1.172	1.264	1.173	1.244	1.179
	1.119	**1.115**	1.213	**1.166**	1.206	**1.157**	1.175	1.204
p=1.2	1.134	**1.212**	1.088	1.190	1.067	1.235	1.112	1.289
	1.148	1.215	1.107	**1.199**	1.051	1.175	1.109	**1.252**
	1.187	1.297	1.133	1.261	1.064	**1.193**	1.061	1.282
p=1.4	1.326	1.386	1.232	1.348	1.176	1.303	1.106	1.311
	1.348	**1.399**	1.260	**1.367**	1.187	1.322	1.073	1.244
	1.365	1.427	1.354	1.446	1.240	**1.427**	1.115	**1.314**
p=1.6	1.535	1.587	1.448	1.541	1.284	1.440	1.329	1.504
	1.559	**1.608**	1.502	**1.579**	1.337	**1.483**	1.210	1.374
	1.577	1.640	1.561	1.690	1.432	1.463	1.393	**1.508**
p=1.8	**1.811**	1.830	1.698	1.755	1.675	1.727	2.190	2.226
	1.813	1.833	1.707	1.763	1.676	**1.749**	1.918	1.664
	1.883	1.893	1.750	**1.831**	1.661	1.680	1.849	**1.767**
p=2	2.078	2.064	2.088	2.067	**2.196**	2.249	2.830	2.941
	2.057	**2.053**	2.047	2.038	2.781	2.744	3.422	3.078
	2.075	2.095	2.009	**2.000**	2.525	2.360	3.165	**2.738**

În tabelul 3.2 sunt afișate erorile medii de reprezentare obținute în timpul rulării cadrelor. Pentru fiecare K, există trei coloane: cea din stânga conține erorile pentru cadrul Half-Adapt; cel din centru contine erorile pentru cadrul Mean-Norm, cel din dreapta contine erorile algoritmilor care sunt in posesia p_{true} pentru fiecare valoare de K si p, cea mai mica eroare de recuperare este scris cu caractere aldine.

Eroarea de reprezentare crește pe măsură ce valoarea lui K crește. FP-GGN și LP_L1 au valori similare pentru erorile de reprezentare, cu diferențe foarte mici. Pe măsură ce K crește, FP-GGN începe să aibă o eroare de reprezentare puțin mai mică în comparație cu LP_L1, OMP-p având cea mai mare eroare.

Table 3.2: Erori medii de reprezentare pentru OMP-p (sus în fiecare celulă), FP-GGN (mijloc) și LP_L1 (jos)

K	5			7			9			11		
	Half-Adapt	Mean-Norm	Fixed	Half-Adapt	Mean-Norm	Fixed	Half-Adapt	Mean-Norm	Fixed	Half-Adapt	Mean-Norm	Fixed
p=1	0.030	0.030	0.029	0.039	0.034	0.041	0.049	0.057	0.062	0.073	0.075	0.091
	0.028	0.028	**0.027**	0.027	0.027	0.026	0.027	0.027	**0.026**	0.026	0.026	**0.025**
	0.028	0.029	0.028	0.027	0.026	**0.025**	0.027	0.027	**0.026**	0.026	0.028	0.026
p=1.2	0.035	0.035	0.032	0.037	0.037	0.036	0.064	0.063	0.058	0.091	0.096	0.083
	0.028	**0.028**	**0.028**	**0.027**	0.028	0.028	**0.027**	**0.027**	0.028	**0.026**	**0.026**	**0.026**
	0.028	0.029	**0.028**	**0.027**	0.029	0.028	**0.027**	**0.027**	**0.027**	0.028	0.031	0.029
p=1.4	**0.029**	**0.029**	**0.029**	0.041	0.041	0.038	0.037	0.036	0.034	0.074	0.077	0.055
	0.029	**0.029**	**0.029**	0.029	0.029	0.029	0.028	0.028	0.028	**0.026**	0.027	0.027
	0.029	**0.029**	**0.029**	0.029	**0.028**	0.029	0.028	**0.027**	0.029	0.028	0.031	0.029
p=1.6	**0.029**	**0.029**	**0.029**	0.031	0.031	0.030	0.042	0.042	0.039	0.066	0.075	0.057
	0.030	0.030	0.030	**0.029**	**0.029**	**0.029**	**0.028**	**0.028**	**0.028**	**0.027**	**0.027**	0.028
	0.030	0.030	0.030	**0.029**	0.030	**0.029**	0.029	0.029	0.029	0.029	0.029	0.028
p=1.8	0.031	0.031	0.031	0.033	0.033	0.034	0.038	0.040	0.043	0.072	0.078	0.073
	0.030	**0.030**	**0.030**	**0.029**	**0.029**	**0.029**	**0.028**	**0.028**	**0.028**	**0.028**	**0.028**	**0.028**
	0.030	**0.030**	**0.030**	**0.029**	**0.029**	**0.029**	0.029	0.029	**0.028**	0.031	0.030	0.029
p=2	0.033	0.033	0.033	0.030	0.030	**0.029**	0.041	0.041	0.038	0.056	0.059	0.049
	0.030	**0.030**	**0.030**	**0.029**	**0.029**	**0.029**	**0.029**	**0.029**	**0.029**	0.030	0.032	**0.029**
	0.030	0.031	**0.030**	**0.029**	**0.029**	0.030	0.030	0.030	**0.029**	0.034	0.036	0.032

Figurile 3.1 conțin informațiile complete pentru toate rulările cu $K = 5$ pentru cadrul Mean-Norm, în timp ce figurile 3.2 conțin informațiile complete pentru toate rulările cu $K = 5$

pentru cadrul Half-Adapt. În aceste cifre, media este afișată cu un simbol diferit; deplasarea orizontală este folosită doar pentru o vizibilitate mai bună; valorile p_{true} sunt aceleași pentru toți algoritmii.

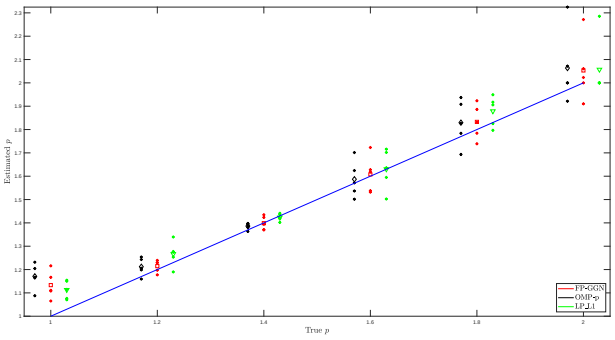

Figure 3.1: Estimarea parametrului de formă pentru zgomot pentru cazul $K = 5$ utilizând algoritmul Mean-Norm

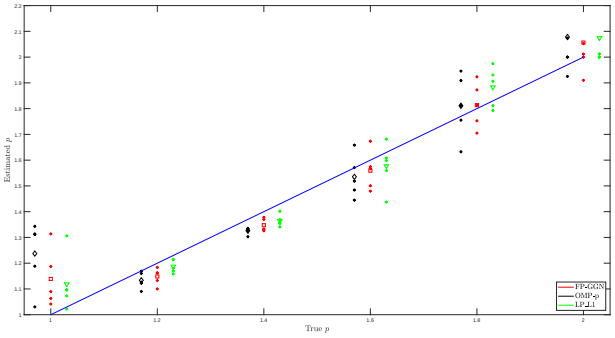

Figure 3.2: Estimarea parametrului de formă pentru zgomot pentru cazul $K = 5$ folosind algoritmul Half-Adapt

Se poate observa că, pe măsură ce K crește, estimarea parametrului de formă tinde să se înrăutățească. Pentru cadrul Mean-Norm, toți algoritmii încep să subestimeze valoarea lui K. În cazul cadrului Half-Adapt, toți algoritmii, de asemenea, arată o tendință de subestimare a valorii ok K. Remarcăm că parametrul de formă este recuperat suficient de bine, în special pentru valorile inferioare ale nivelului de dispersie K. Cele mai slabe aproximări apar atunci când $p_{\text{true}} = 1$; o cauză care contribuie este faptul că valorile $p < 1$ nu sunt posibile, deci o anumită părtinire inerentă.

Erorile algoritmilor adaptivi sunt aproape de cele ale algoritmilor cunoscând p_{true}, uneori mai bune; deci, cadrul propus de noi este capabil să ofere reprezentări bune, chiar dacă p_{true} este necunoscut. FP-GGN oferă o recuperare mai bună a erorilor în majoritatea cazurilor. Cadrul Mean-Norm oferă o eroare de recuperare medie puțin mai bună decât cadrul Half-Adapt și oferă o estimare mai bună pentru parametrul de formă.

3.2 Aplicație pentru imagini

Considerăm o imagine $Y \in \mathbb{R}^{t \times t}$ care este perturbată de un semnal gaussian generalizat $V \in \mathbb{R}^{t \times t}$ cu medie zero și o abatere standard $\sigma = 1$, care se obține prin eșantionarea unei distribuții cu

parametrul de formă p. Considerăm cazul zgomotului aditiv, iar imaginea perturbată $\widetilde{Y} \in \mathbb{R}^{t \times t}$ se obține cu

$$\widetilde{Y} = Y + V \tag{3.3}$$

Folosind un număr mic de patch-uri $P \in \mathbb{R}^{s \times s}$, cu $s \leq t$ și $m = s \times s$, din imaginea perturbată $\widetilde{Y}$, un dicționar D este antrenat. După ce dicționarul este obținut din imaginea perturbată, sunt selectate mai multe patch-uri P și se obține un semnal de reprezentat Y^*. Problema de optimizare care rezultă este

$$\begin{aligned} \underset{X \in \mathbb{R}^{n \times r}}{\text{minimize}} \quad & \|Y^* - DX\|_p \\ \text{subject to} \quad & \|X_i\|_0 \leq K, i \in \{1, ..., q\} \end{aligned} \tag{3.4}$$

unde q este numărul de patch-uri utilizate pentru problema de curățare a zgomotului.

Problema (3.4) poate fi reformulată ca o problemă LASSO, care ne oferă programul de optimizare

$$\underset{X \in \mathbb{R}^{n \times q}}{\text{minimize}} \quad \|Y^* - HX\|_p + \lambda \|X\|_1 \tag{3.5}$$

În practică, pentru fiecare patch, se rezolvă problema de optimizare (2.2). Imaginea este apoi reconstruită, folosind patch-urile obținute Y^*.

Ideea este că, după finalizarea procesului de învățare, în dicționarul D există atomi care reprezintă cel mai mult imaginea neperturbată și atomi care reprezintă cel mai mult zgomotul. Când imaginea este reconstruită folosind dicționarul rar obținut D, atomii K selectați vor conține majoritatea informațiilor lor din imaginea neperturbată.

Cadrul de curățare de zgomot al imaginii, care poate fi derivat, este prezentat în algoritmul 3.2, unde la fiecare pas de reprezentare, fiecare patch este calculat folosind dicționarul antrenat D. Folosind reprezentarea obținută a patch-urilor X, imaginea este reconstruită și se obține o nouă imagine Y_{rec}. Algoritmul 3.1 va fi utilizat cu algoritmul OMP-p.

Ambele variante ale acestui algoritm, Half-Adapt și Mean-Norm sunt testate și comparate cu OMP și OMP-p cu valoarea cunoscută de p_{true}. Se poate observa că Half-Adapt are cele mai bune valori pentru PSNR și SSIM, în comparație cu toți algoritmii. Mean-Norm este mai bun decât OMP, estimează mai bine parametrul de formă p decât Half-Adapt și este mai rapid decât Half-Adapt. Se poate observa că timpul de rulare și calitatea estimării depind de actualizarea parametrului de formă utilizată.

Dicționarul este antrenat folosind 2560 de patch-uri din imaginea perturbată, folosind o procedură de învățare a dicționarului care utilizează OMP-p. Un alt set mai mare de 15876 de patch-uri este apoi extras și este folosit pentru procesul real de dezgomot al imaginii.

Dimensiunea selectată pentru dicționare este de 64×128. Patch-urile au dimensiunea 8×8. Nivelul de dispersie este $K = 8$. Parametrii de formă sunt $p \in \{1.2, 1.4, 1.6, 1.8\}$. Valorile SNR sunt alese ca 20. Testele se fac pe trei imagini: Lena, Barbara și Boat. Din fiecare imagine, este selectat un patch mare de 256×256 din care sunt extrase patch-urile de antrenament și de învățare. Fiecare imagine este perturbată de zgomot obținut prin eșantionarea zgomotului Generalizat Gaussian cu parametru de formă p. Pentru fiecare combinație de imagine și valoare de p se fac 5 teste.

Algorithm 3.2: Algoritm pentru curățarea zgomotului

Data: Set de patch-uri de reprezentat $Y \in \mathbb{R}^{m \times q}$, dictionar $D \in \mathbb{R}^{m \times n}$, nivel de sparsitate $K \in \mathbb{Z}$, număr maxim de iterații pentru estimarea normei ℓ_p $Iter_{norm}$, limită de oprire θ

Result: Reprezentare rară $X \in \mathbb{R}^{n \times q}$, parametrul de formă estimat $p \in \mathbb{R}$, $p \geq 1$

1 Antrenați dicționarul folosind orice algoritm care optimizează folosind norma p
2 Calculați setul X folosind algoritmul cu $p = 2$ pentru fiecare coloană de Y
3 Utilizați algoritmul din [8] pentru a estima parametrul de formă $\tilde{p}$ din erorile de reprezentare
4 Actualizați norma p folosind (3.1) sau (3.2)
5 **while** *număr de iterații* $\leq Iter_{norm}$ *sau* $|p - \tilde{p}| > \theta$ **do**
6 Calculați setul de reprezentare rară X pentru fiecare coloană de Y folosind algoritmul 3.1 cu p de la pasul anterior
7 Utilizați algoritmul în [8] pentru a estima parametrul de formă $\tilde{p}$
8 Actualizați norma p folosind (3.1), pentru varianta Half-Adapt, sau (3.2), pentru varianta Mean-Norm
9 **end**
10 Dacă este utilizată varianta Mean-Norm și numărul de iterații a fost mai mare decât 1, calculați norma p ca valoare medie a tuturor valorilor anterioare ale lui $\tilde{p}$.
11 Calculați setul de reprezentare rară X folosind algoritmul cu p obținut.
12 Patch-urile rezultate sunt folosite pentru a reconstrui imaginea calculând pentru fiecare pixel al imaginii media pentru toate patch-urile care conțin acel pixel.

În figura 3.3, sunt prezentate imaginea originală (în stânga), imaginea perturbată (în centru) și imaginea dezgomotată (în dreapta) obținute folosind varianta Half-Adapt a algoritmului, pentru a arăta cum este modificată imaginea, când este procesată folosind cadrul 3.2. Este evident că calitatea imaginii este îmbunătățită și cea mai mare parte a zgomotului este eliminată din imaginea perturbată.

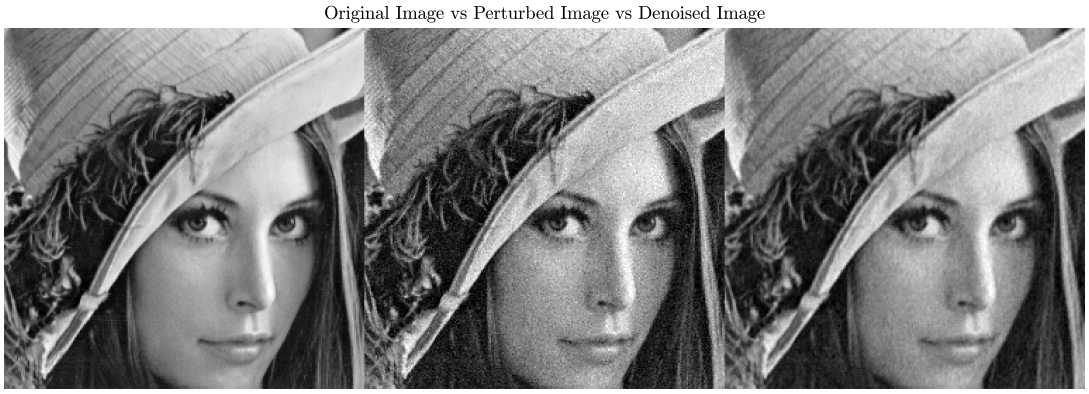

Figure 3.3: Curățarea imaginilor pentru $K = 1, 2$, pentru algoritmul Half-Adapt pentru imaginea de testare Lena

Tabelul 3.3 arată estimarea valorii medii a parametrului de formă pentru versiunea Mean-Norm și Half-Adapt a algoritmului 3.2. Cu aldine, se scrie estimarea care se apropie cel mai mult de valoarea reală. Se poate observa că versiunea Half-Adapt tinde să subestimeze parametrul de formă, în timp ce versiunea Mean-Norm tinde să supraestimeze. Pentru $p = 1.2$, estimarea Half-Adapt a parametrului este cea mai apropiată de valoarea reală a p, în timp ce pentru $p = 1.4$ și $p = 1.6$ estimările Mean-Norm sunt mai apropiate de valoarea reală, în multe valoarea testată fiind estimată la 2. Pentru $p = 1, 8$, ambele versiuni estimează parametrul de formă la 2.

Table 3.3: Parametrii de formă medii pentru Mean-Norm și Half-Adapt

	Mean-Norm	Half-Adapt
p=1.2	1.498	**1.154**
p=1.4	**1.576**	1.177
p=1.6	**1.708**	1.417
p=1.8	**2**	**2**

În tabelul 3.4, valorile medii PSNR sunt prezentate pentru algoritmii OMP, OMP-p, Mean-Norm și Half-Adapt. Pentru fiecare valoare de p, cele mai mari valori PSNR sunt scrise cu caractere aldine. Mean-Norm are un PSNR mai bun decât OMP, în timp ce are o valoare mai mică decât OMP-p. Half-Adapt are cea mai bună valoare în toate cazurile. Pentru $p = 1,8$ Mean-Norm și Half-Adapt au aceeași valoare ca OMP, deoarece parametrul de formă este estimat cu valoarea 2.

Table 3.4: PSNR mediu pentru OMP, OMP-p, Mean-Norm and Half-Adapt

	OMP	OMP-p	Mean-Norm	Half-Adapt
p=1.2	29.629	29.882	29.791	**29.897**
p=1.4	29.593	29.731	29.703	**29.765**
p=1.6	29.620	29.640	29.640	**29.687**
p=1.8	**29.737**	29.733	**29.737**	**29.737**

Valorile medii ale SSIM sunt prezentate în tabelul 3.5, cu cea mai bună valoare scrisă cu caractere aldine. Comportamentul este ca pentru valorile PSNR. Mean-Norm este mai bun decât OMP, dar are o valoare mai mică decât ceilalți algoritmi. Half-Adapt se dovedește a avea încă o dată cel mai bun rezultat. Pentru $p = 1,8$, toți algoritmii au același rezultat, deoarece ambele variante ale algoritmului 3.2 estimează parametrul de formă la 2.

Table 3.5: Indicele de similitudine structurală medie pentru OMP, OMP-p, medie-normă și semi-adaptare

	OMP	OMP-p	Mean-Norm	Half-Adapt
p=1.2	0.794	0.812	0.804	**0.814**
p=1.4	0.795	0.804	0.801	**0.809**
p=1.6	0.795	0.799	0.798	**0.802**
p=1.8	**0.800**	**0.800**	**0.800**	**0.800**

În medie, timpul de rulare pentru 15876 patch-uri, este de 5097 secunde pentru Half-Adapt, în timp ce pentru Mean-Norm timpul de rulare este de 3779 secunde, pentru OMP-p cu p_{true} timpul de rulare este de 751 secunde și pentru OMP timpul de rulare este de 0,002 secunde. Se poate observa că Mean-Norm este mai rapid decât Half-Adapt. Timpul de rulare crește pe măsură ce scăderea valorii parametrului de formă estimat este atenuată.

Pentru toate cazurile, se poate observa că Half-Adapt oferă cea mai bună valoare a PSNR și SSIM.

Bibliography

[1] T. Achterberg and T. Berthold. "Improving the feasibility pump". In: *Discrete Optimization* 4.1 (2007), pp. 77–86.

[2] A. Beck and M. Teboulle. "A fast iterative shrinkage-thresholding algorithm for linear inverse problems". In: *SIAM journal on imaging sciences* 2.1 (2009), pp. 183–202.

[3] L. Bertacco, M. Fischetti, and A. Lodi. "A feasibility pump heuristic for general mixed-integer problems". In: *Discrete Optimization* 4.1 (2007), pp. 63–76.

[4] N. Boland et al. "Boosting the feasibility pump". In: *Mathematical Programming Computation* 6.3 (2014), pp. 255–279.

[5] S. Bourguignon et al. "Exact sparse approximation problems via mixed-integer programming: Formulations and computational performance". In: *IEEE Transactions on Signal Processing* 64.6 (2015), pp. 1405–1419.

[6] S. Dey et al. "Exploiting sparsity of MILPs by improving the randomization step in feasibility pump". In: *SIAM Journal on Optimization,(to appear)* (2016).

[7] S.S. Dey et al. "Improving the randomization step in feasibility pump". In: *SIAM Journal on Optimization* 28.1 (2018), pp. 355–378.

[8] J. Dominguez-Molina et al. *A practical procedure to estimate the shape parameter in the generalized Gaussian distribution*. Available at `http://www.cimat.mx/reportes/enlinea/I-01-18_eng.pdf`. 2003.

[9] M. Fischetti, F. Glover, and A. Lodi. "The feasibility pump". In: *Mathematical Programming* 104.1 (2005), pp. 91–104.

[10] M. Fischetti and D. Salvagnin. "Feasibility pump 2.0". In: *Mathematical Programming Computation* 1.2-3 (2009), pp. 201–222.

[11] B. Geißler et al. "Penalty alternating direction methods for mixed-integer optimization: A new view on feasibility pumps". In: *SIAM Journal on Optimization* 27.3 (2017), pp. 1611–1636.

[12] M. Grant and S. Boyd. *CVX: Matlab Software for Disciplined Convex Programming, version 2.1*. http://cvxr.com/cvx. 2014.

[13] K. Huang and S. Mehrotra. "An empirical evaluation of walk-and-round heuristics for mixed integer linear programs". In: *Computational Optimization and Applications* 55.3 (2013), pp. 545–570.

[14] A. Lodi. "Mixed integer programming computation". In: *50 years of integer programming 1958-2008*. Springer, 2010, pp. 619–645.

[15] M. Rasmussen and R. Bro. "A tutorial on the Lasso approach to sparse modeling". In: *Chemometrics and Intelligent Laboratory Systems* 119 (2012), pp. 21–31.

[16] J. Tropp. "Greed is good: Algorithmic results for sparse approximation". In: *IEEE Transactions on Information theory* 50.10 (2004), pp. 2231–2242.

[17] L. Wang, M. D Gordon, and J. Zhu. "Regularized least absolute deviations regression and an efficient algorithm for parameter tuning". In: *6th Int. Conf. Data Mining*. IEEE. 2006, pp. 690–700.

[18] F. Wen et al. “Robust Sparse Recovery in Impulsive Noise via ℓ_p-ℓ_1 Optimization”. In: *IEEE Transactions on Signal Processing* 65.1 (2016), pp. 105–118.

[19] W.J. Zeng, H.C. So, and X. Jiang. “Outlier-Robust Greedy Pursuit Algorithms in ℓ_p-Space for Sparse Approximation”. In: *IEEE Trans. Signal Processing* 64.1 (2016), pp. 60–75.