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

# PHD THESIS

# Abstract - Feasibility Pump Adaptations for Sparse Representations

Absolvent
Miertoiu Florin Ilarion

Advisor
Prof. dr. ing. Bogdan Dumitrescu

Bucharest, 2022

# 1 Introduction

The sparse representation of a signal $y \in \mathbb{R}^q$, using a dictionary $D \in \mathbb{R}^{m \times n}$, consists in finding a solution $x \in \mathbb{R}^n$ having the smallest possible number of non-zero coefficients to the system $y = Dx$. Each column of the dictionary $D$ is considered an atom that can take part in the sparse representation $x$ of signal $y$. If the corresponding value of $x$ for an atom of dictionary $D$ is not null, than that atom is part of the support of the sparse representation $x$ of the signal $y$. In this case the system is underdetermined as $m \leq n$. Also the number of atoms used in the representation is much smaller than the number of columns $n$.

The optimization that is considered in this case is

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

Problem (1.1) is NP-hard and in practice is modified in order for linear optimization algorithm to be used. The equality constraint is replaced by the misfit measure $\|y - Dx\|_p \leq u$ where the used norm is convex ($p \geq 1$), in multiple cases $p = 2$. Since the norm 0 problems are NP-hard, in multiple problems it is replaced with the 1 norm which results in problems which can be solved in polynomial time.

The most used adaptation for problem (1.1) in application is the Least Absolute Shrinkage and Selection Operator (LASSO) model [15] which is defined as

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

In this model, each column of the matrix $D$ is viewed as a feature and the vector to be represented $y$ is viewed as the output of a complex system. The model tries to find a small subset of the features which allow for a precise representation of $y$.

Mixed Integer Programs are optimization problems that contain both integer and continuous variables. If all the variables are integers, the problem is a pure integer problem.

The general form of a mixed integer programming is [14]

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

with $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. $I$ is the set that contains the indices of the integer variables in $x$.

The original problem (1.3) is NP-hard and computationally expensive to solve.

The Feasibility Pump, initially proposed in [9] and [3], is a MIP heuristic method that generates two sequences of solutions, one satisfying the linear constraints and one satisfying the integer constraint. These sequences are generated until a solution that satisfies both conditions is found. The Feasibility Pump starts from an initial solution and then proceeds, through several iterations, to minimize the distance between the two solutions, first by solving a linear optimization problem with the integer constraints removed in order to obtain the solution that satisfies the linear constraint, then by a rounding step in order to obtain the solution that satisfies the integer constraint. The algorithm is prone to cycling and loops and several approaches have been proposed to avoid this [7, 11, 6, 13, 4]. The Feasibility Pump offers much better execution time (several orders of magnitude compared to other MIP algorithms), but the quality of the solution is not as good; there is no guarantee that the optimum is attained.

We consider the MIP problem

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

with A being an $m \times n$ matrix and $I \subseteq \{1, 2, ..., n\}$ is the index set of the integer variables. In order to get a linear problem, the integer condition in (1.4) for the subset $I \subseteq \{1, 2, ..., n\}$ is relaxed to:

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

We consider a solution $x$ for the relaxed problem (1.4) to be integer, if all elements $x_j$ are integer for all $j \in I$. Using a rounding procedure, an integer vector $\tilde{x}$ is obtained from the relaxed solution $x$. We define the operator that computes the difference between the two solutions as

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

The operator (1.6) is used in order for force the solution of the relaxation of problem (1.4) to be closer to the integer solution $\tilde{x}$. This operator can be reformulated as in [1], for general mixed integer problems, as

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

where $d_j = |x_j - \widetilde{x}_j|$ for the integer variables $x_j$ that are not rounded to any of the two bounds. This term that does not appear in the binary case. In this case (1.7) becomes [9]

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

The Feasibility Pump, which was first proposed in [9], uses as a starting point a feasible solution of the relaxed problem (1.4), from which it creates two sequences of points $x$ and $\tilde{x}$, where $x$ is the solution of the relaxed problem, but does not necessary respect the integer constraint and $\tilde{x}$ which respects the integer condition, but it is not necessary a solution of the relaxed problem. In order to generate the two sequences, the following schema is applied: at each iteration (which in this algorithm is also called a pumping cycle), a new integer solution $\tilde{x}$ is obtained from the relaxed solution $x$ by simply rounding the integer-constrained component to the nearest integer or a different procedure depending on the problem, while a new relaxed solution $x$ is obtained by minimizing the relaxed problem, defined by:

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

This procedure is applied until an integer $x$ is found or an iteration limit is reached. As this scheme is prone to cycling, a perturbation step is necessary [10].

The Feasibility Pump algorithm steps are displayed in Algorithm 1.

**Algorithm 1.1:** Feasibility Pump

```
   Data: $MIP = min\{c^T x : x \in P, Ax \le b, l \le x \le u, x_j$ integer $\forall j \in I\}$
   Result: a feasible MIP solution $\widetilde{x}$ (if found)
1  $x = \text{argmin}\{c^T x : Ax \le b, x \in P\}$;
2  while not termination condition do
3      if $x$ is integer then
4          return $x$ ;
5      else
6          $\widetilde{x} = Round(x)$;
7      end
8      if cycle is detected then
9          $Perturb(\widetilde{x})$;
10     end
11     $\bar{x} = \text{argmin}\{\triangle(x,\widetilde{x}) : Ax \le b, l \le x \le u, x \in P\}$ ;
12 end
```

The algorithm tries to solve, at step 1, the initial relaxed problem of (1.4). The Feasibility Pump uses, at step 11, the relaxed problem (1.9) for each iteration. The solution to the relaxed problem is verified if it satisfies the stopping criteria of the algorithm at step 3.

If the solution of the relaxed problem does not satisfy the stopping criteria, a rounding procedure, step 6, is used to obtain the integer solution that will be used for the next Feasibility Pump iteration.

The algorithm is prone to cycling. If a cycle is detected, the integer solution is perturbed at step 8. Multiple perturbation strategies have been considered depending on the type of problem used.

As the initial problem is not present in each pumping cycle, the solution quality is not very good as the algorithm is focused more on finding the integer solution. In order to solve this issue, the Objective Feasibility Pump was proposed in [1]. It introduces the initial problem in the pumping cycles. With each iteration, the influence of the initial problem is reduced in order to allow the algorithm to converge. With this modification problem (1.9) is replaced by

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

The reduction of the influence of the initial problem $\alpha$ is done by multiplication with a value $\gamma \in (0,1)$ after each Feasibility Pump step:

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

By adding the above modifications, the Objective Feasibility Pump Algorithm 2 is obtained.

**Algorithm 1.2:** Objective Feasibility Pump

```
Data: $MIP = \min\{c^T x : x \in P, Ax \leq b, l \leq x \leq u, x_j$ integer $\forall j \in I\}$
Result: a feasible MIP solution $\widetilde{x}$  (if found)
$x = \text{argmin}\{c^T x : Ax \leq b, x \in P\}$;
while not termination condition do
    if $x$ is integer then
        return $x$ ;
    else
        $\widetilde{x} = Round(x)$;
    end
    if cycle is detected then
        $Perturb(\widetilde{x})$;
    end
    $\bar{x} = \text{argmin}\{(1-\alpha)\triangle(x, \widetilde{x}) + \alpha(c^T x) : Ax \leq b, l \leq x \leq u, x \in P\}$ ;
    $\alpha \leftarrow \gamma\alpha$ ;
end
```

The addition of the objective in the pumping cycles and the reduction parameter $\alpha$ require a modification in the cycling detection of the algorithm. It can happen that, during the first pumping cycles, the same integer solution is found. At each step of the algorithm, the integer solution $\tilde{x}$ and $\alpha$ are stored in pairs. The restart or perturbation procedure is then applied, only if at a specific step $t$ we find that $\alpha_t - \alpha_{t-1} \geq \delta$, where $\delta \in [0,1]$ is set at the start of the algorithm [1].

# 2 Feasibility Pump Adaptations for Perturbed Signals

## 2.1 Feasibility Pump Adaptation for Sparse Representations

The initial sparse representation problem considered to be solved using the Feasibility Pump is

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

where the data misfit measure $\|y - Dx\|_p$ can use any norm $p$.

The form of the problem, that will be used as a starting point for the adaptations of the Feasibility Pump, is the LASSO style-problem

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

which tries to balance the two quantities that need to be minimized, the misfit measure and the number of atoms in the support.

In order for the Feasibility Pump algorithm to be used, a binary variable $b \in \mathbb{Z}^q$ is introduced, which shows if an atom of the dictionary is used to represent the signal $y$.

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

In order to create the model for the Feasibility Pump steps, problem (2.2) is combined with the general form of the Objective Feasibility Pump problem (1.10). The problem that is solved at each pumping step becomes

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

where $\alpha$ is the decay parameter, used in the Objective Feasibility Pump formulation in order to increase the importance of the integer condition compared to the initial problem. The reduction of $\alpha$ is done by multiplication with a value $\gamma \in (0, 1)$ as in (1.11).

The term $\beta$ is used to counterbalance the value of the regularization term $\|x\|_1$, when compared to the misfit term.

The big-$M$ trick is used, as in paper [5], where $M$ is a preset parameter bounding the value of $x$, chosen as $M = 1.1\|H^T y\|_\infty / \|H\|_2^2$. It limits the amplitude of the atoms and helps with obtaining a large number of smaller coefficients which are close to 0.

The general form of the algorithm is similar to algorithm 2, with the initial problem selected being (2.2) and the pumping cycles using (2.4).

**Algorithm 2.1:** Modified Feasibility Pump

```
Data: Dictionary $D \in \mathbb{R}^{m \times n}$, signal to represent $y \in \mathbb{R}^m$, number of non-zero coefficients
      used for the representation $K \in \mathbb{Z}$, maximum number of iterations $Iter$, weight
      parameters $\alpha$, $\gamma$
Result: a feasible solution $x \in \mathbb{R}^n$
Solve (2.3) for $b \in \mathbb{R}^n$ using an optimization solver. The vectors $x$ and $b$ are obtained.
Use rounding procedure to obtain vector $\widetilde{b}$.
while number of iterations $\leq Iter$ do
    Solve the problem (2.8). The vectors $x$ and $b$ are obtained.
    if $b$ is integer then
        return $x$;
    end
    Use rounding procedure to obtain vector $\widetilde{b}$.
    if cycle is detected then
        Use selected perturbation strategy on $\widetilde{b}$.
    end
    Update the value of $\alpha$ using (1.11).
end
```

For the rounding procedure, the real $b$ is rounded in order to obtain a binary vector $\widetilde{b}$, by setting the $K$ values of $b$ corresponding to the $K$ largest amplitudes of $x$ to 1 and the others to 0. This procedure is used for all variants of the Feasibility Pump presented in this thesis.

The perturbation used is a weak perturbation of the form

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

which is used on the integer vector $\widetilde{b}$ after a certain condition is fulfilled.

## 2.2 Laplacian Noise Case

For the Laplacian perturbation case the equality condition $y = Dx$ is relaxed and replaced by the minimization of the data misfit measure $\|y - Dx\|_1$. The number of non-zero coefficients is bounded by the threshold $K$ and the sparse representation is found by solving the problem

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

In order for the Feasibility Pump algorithm to be used, we introduce a binary variable $b \in \mathbb{Z}^n$, which shows whether an atom of the dictionary is used to represent the signal $y$. Each column of the dictionary $D$ is associated with an element of the vector $b \in \{0; 1\}^n$. An element in $b$ that has a value 0 shows that the respective atom of the dictionary does not participate in the representation, while an element with value 1 shows that that atom is used for representing the signal $y$.

The problem (2.6) becomes

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

In order to create the model for the pumping steps, problem (2.7) is combined with the general form of the Objective Feasibility Pump problem (1.10). The problem that is solved at each pumping step becomes

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

The version of Feasibility Pump for the Laplacian noise case is implemented in MATLAB using the *linprog* and compared to the regularized least absolute deviation (RLAD) model [17] and the Branch and Cut algorithm. For the model 2.7 two variants are considered, the one which uses the regularization term (SFPreg) and the one without it (SFP). It is proven that the regularization term helps the algorithm have better results. Also the early start is considered in order to prove that it can improve the running time with minimal performance reduction.

For our tests, we use randomly generated dictionaries with condition numbers 10, 100, 1000, 10000 and 100000. For each condition number, 160 dictionaries of two different sizes, 80 × 200 and 50 × 100, are generated. The test signals are obtained with $y = Dx_{\text{true}} + u$, where the solutions $x_{\text{true}}$ have $K \in \{5, 7, 9, 11\}$ nonzero coefficients generated randomly following a Gaussian distribution, in random positions; the noise $u$ is Laplacian and its variance is chosen such that the signal to noise ratios have values $10, 20, 30,\ \infty$.

For the computation of the representation error, the relative error

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

is used (where now $x$ is the computed solution).

The relative recovery errors is computed using

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

where $x_{true}$ is the true representation of the signal $y$ using the dictionary $D$ for the unperturbed case.

We have implemented SFP and SFPreg as shown in Algorithm 2.1. We also consider a starting weight value $\alpha = 1$. At each iteration $\alpha$ is then multiplied by an update factor $\gamma = 0.95$. The number of iterations $Iter$ is set to 1000. We run SFPreg with 50 values of the regularization parameter $\lambda$ from 0 to 1. The value for which the error is the smallest is considered the best value.

The two variants of the Feasibility pump algorithm are compared with the regularized least absolute deviation (RLAD) model [17]

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

solved with *linprog*. Note that this is a lasso-style relaxed problem, adapted to the $\ell_1$ error norm.

The algorithms are implemented in MATLAB and tested on a computer with a 6-core 3.4 GHz processor and 32 GB of RAM.

First we compare the error difference, computed using (2.9) between the two implementations presented, SFP and SFPreg. In figures 2.1 it can be seen that in most cases SFPreg has a smaller error than SFP. It is obvious that SFPreg is a better algorithm for having a representation with a smaller error than SFP. Both SFP and SFPreg recover without error in the case where $SNR = \infty$.

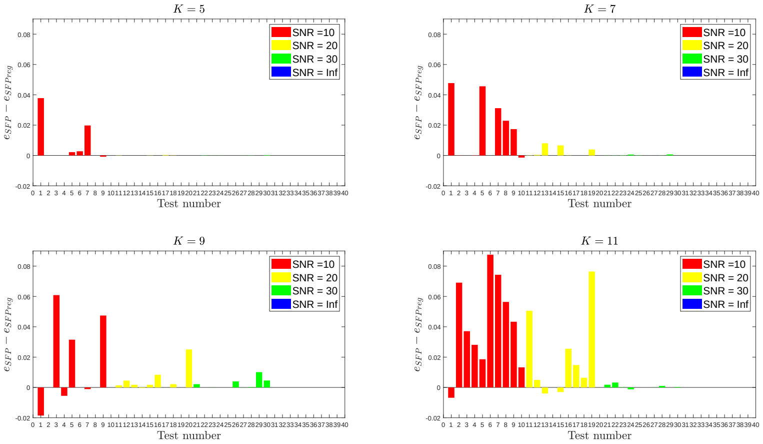

Figure 2.1: Mean representation error difference between SFP and SFPreg for a dictionary conditioning of 10

In figures 2.2 the difference between the mean recovery errors computed using (2.10) is displayed. The behavior is similar as in the case of the representation error, with SFPreg being better than SFP in most cases.

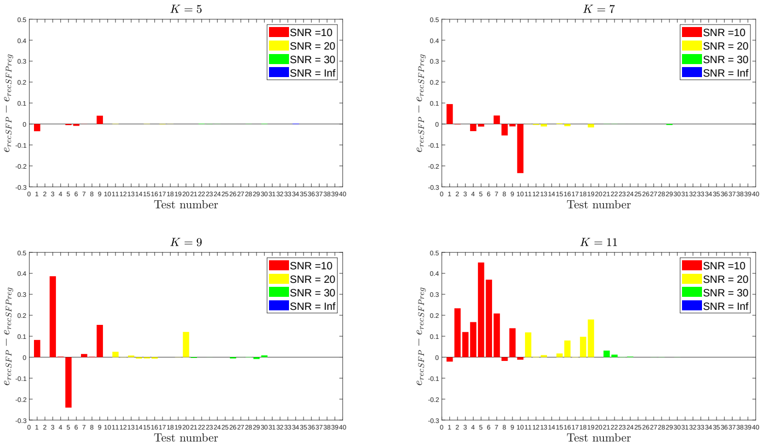

Figure 2.2: Mean recovery error difference between SFP and SFPreg for a dictionary conditioning of 10

It can be seen that the representation error difference is much smaller than the recovery error difference, which indicates that SFP has more false positives in the support of the solution than SFPreg. In the unperturbed case, both algorithms recover the true support. In the perturbed cases, SFPreg has 1.32 false positives per case while SFP has 1 false positive per case. SFP recovers the true support in 294 cases out of 600 cases (49%), while the SFPreg recovers the true support in 317 out of 600 cases (53%). A false negative indicate when an atom is missing from the support of the true solution. A false positive appears when the support of the computed solution contains atoms outside the true support.

SFP requires 0.5 seconds to run the unperturbed case and 3.6 seconds for the perturbed case, while SFPreg requires 0.6 seconds in the unperturbed case and 1.8 seconds for the perturbed case. The addition of the regularization parameter improves the running time of the algorithm. This can also be observed for the number of iterations used for the perturbed cases, SFPreg using 5.4 iterations and SFP needing 16.2 iterations in average.

RLAD finds the true support in all the unperturbed cases, but in the perturbed case it finds the true support in only 46 of the 600 (7.7%) perturbed cases. RLAD indicates 3.03 false negatives per pertubed case. Also RLAD shows false positives in 543 out of 600 (90.5%) perturbed cases, with an average of 0.9 false positives per case. RLAD recovers the support just as well as SFP and SFPreg in the unperturbed case, but it underperforms both algorithms in the perturbed cases.

In average, Branch and Bound implementation requires 165 seconds to run an unperturbed case and 360 seconds for the perturbed case. It recovers the true support in all the unperturbed cases and in 336 out of 600 cases (56%). Branch and Bound indicates in average 0.87 false negatives per case, in the perturbed cases. Branch and Bound outperforms slightly when compared with the SFPreg, but the running time is larger by three orders of magnitude.

Table 2.1: Mean Representation Errors for the algorithms for the condition number 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 | $6.575 \cdot 10^{-13}$ | $3.116 \cdot 10^{-10}$ | $2.599 \cdot 10^{-13}$ | $3.829 \cdot 10^{-10}$ | $\mathbf{7.676 \cdot 10^{-16}}$ |

The SFPreg algorithm offers better error recovery that RLAD and SFP. It also seems to offer a good compromise, when compared to the Branch and Bound implementation, as it offers similarly good support recovery, but with a much better execution time.

## 2.3 Gaussian Noise Case

In order to obtain the sparse representation $x$ of a signal $y$ affected by a Gaussian perturbation, in equation (2.1) the data misfit measure $\|y - Dx\|_2$ is used. The number of non-zero coefficients is bounded similarly by the threshold $K$ and the sparse representation is found by solving the problem

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

We propose to combine the MIP approach with the lasso problem (2.12). The binary variable $b \in \{0,1\}^n$ is introduced to perform the role of an indicator that shows which atom of the dictionary $D$ is used for the representation of $y$. We then combine (**??**) with (2.12) to obtain the problem

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

which will be the problem formulation that will be analyzed in this chapter and will be used for the modifications of the Feasibility Pump algorithm.

In order to implement problem (2.13), an auxiliary variable $w \in \mathbb{R}^n$ is introduced for the regularization term, resulting in

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

In each iteration of Feasibility Pump, at step 4, the vector $b$ and the tentative solution $x$ are updated by solving

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

where $\triangle(b,\widetilde{b}) = \|b - \widetilde{b}\|_1$ and $err_{init}$ is the representation error at step 1 of the algorithm **??**. This quadratic programming problem is also solved with quadprog in MATLAB.

During our tests, it was observed that the addition of the term $\frac{K}{err_{iter}^2}$ in (2.15) is necessary to increase the influence of the error in the optimization process for a longer period of time during the test runs, because it is much smaller than the $\triangle(b,\widetilde{b})$ and $\|x\|_1$ terms and it needs an additional weight parameter.

This version of the Feasibility Pump is compared with various algorithms designed for signals affected by Gaussian Noise. The algorithms used for comparing are the OMP (Orthogonal Matching Pursuit) [16], the FISTA (Fast Iterative Shrinkage-Thresholding Algorithm) and the ADMM modification for the Feasibility Pump presented in [11]. It is shown that the Feasibility Pump offers better results in most cases when compared with the other algorithms.

For the computation of the representation error a relative error, which uses the $l_2$ norm is used

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

is used (where now $x$ is the computed solution).

We have implemented the version for the Gaussian noise case, named SQFP.. The initial weight is set as $\alpha = 1$ and is multiplied by an update factor $\gamma = 0.9$ at each iteration; these values seem to provide a good compromise between convergence speed and representation error. We use $\gamma = 0.9$ in this chapter, not $\gamma = 0.95$, because the model is very slow with the addition of $\frac{K}{err_{iter}^2}$ and it allows the speed of our simulation to be better, with the loss in precision being small. The number of iterations $Iter$ is set to 1000. We run SQFP with 50 equally spaced values of the regularization parameter $\lambda$ from 0 to 1. The value for which the mean representation error is the smallest and is considered the best choice for $\lambda$. The dictionaries have been generated the same as in the Laplacian case.

The algorithms that are used for comparison are OMP (Orthogonal Matching Pursuit) [16], FISTA (Fast Iterative Shrinkage-Thresholding Algorithm) [2] and FP-ADM [11].

The mean errors, obtained by running the tests, are displayed in figures 2.3. The first (red) bar in each cell corresponds to the relative error of FISTA, the second (green) is for SQFP, the third (blue) is for OMP and the last (yellow) is the FP-ADM. It can be seen that as the sparsity level $K$ increases, the SQFP algorithm has a much smaller representation error than FISTA and, only for some condition numbers, than OMP and FP-ADM. For $K = 5$ and $K = 7$, the difference between the algorithms is very small. For larger $K$, SQFP shows that it is clearly the better algorithm.

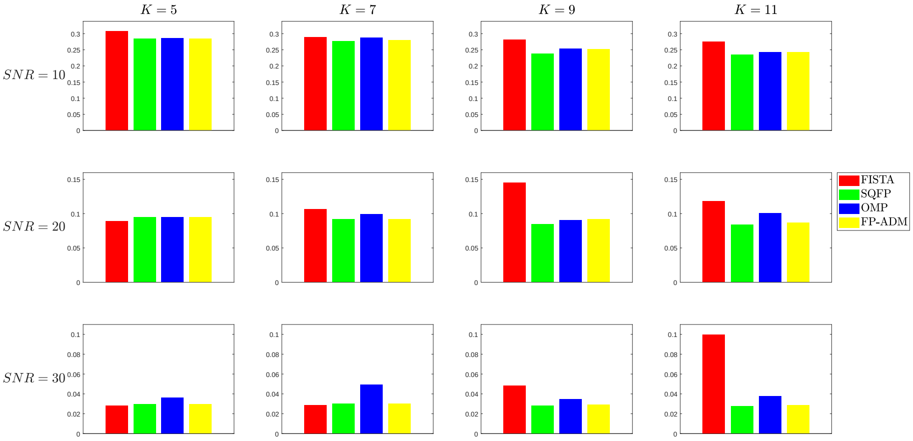

Figure 2.3: Mean Errors for the algorithms tested for a dictionary conditioning of 10

The mean recovery behavior of the four algorithms is presented in figures 2.4. Similarly to the mean error behavior, the SQFP has a much better recovery error than the FISTA algorithm and in multiple cases is better than OMP and FP-ADM. Once again FP-ADM follows the behavior of SQFP, but with a larger error.

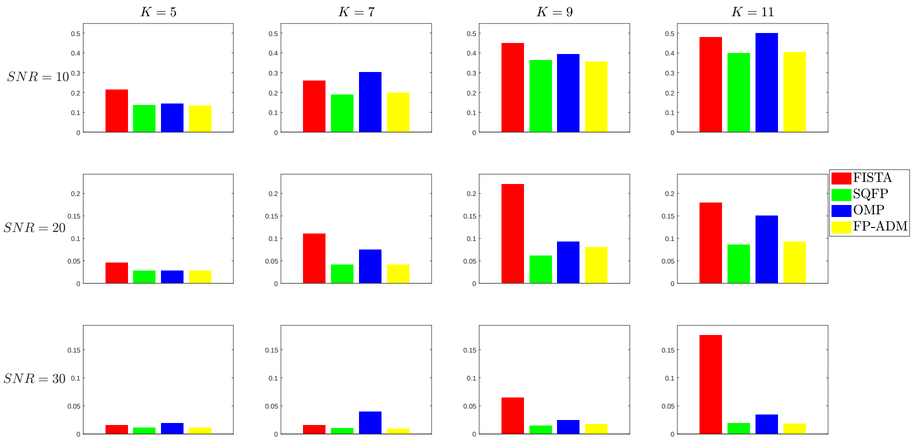

Figure 2.4: Mean Recovery Errors for the algorithms tested for a dictionary conditioning of 10

In tables 2.2 the mean errors of the algorithms are displayed for each $K$ and SNR, for the condition numbers 10. It can be seen that in the non-perturbed representation signal case only SQFP manages to recover the signal in all cases. It can be seen that all other algorithms have instances in which they don't recover the unperturbed signal.

Table 2.2: Mean Error for the algorithms for the condition number 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 |

In average, the number of iterations of SQFP, is around 4.29 iterations for the unperturbed cases and 22.52 for the perturbed cases. Also, the perturbation step appears only in 6 of the 600 cases. The true support is perfectly recovered for all the unperturbed cases and in the perturbed case the average number of atoms, that are missed from the support, is 1.32, with the support being recovered correctly in 305 of the 600 perturbed cases (51%). The means running times are 0.64 seconds for the unperturbed case and 2.4 seconds for the perturbed case. SQFP also finds, in average, 0.94 false negatives in the perturbed case.

FISTA requires, in average, 1352 iterations for the unperturbed case and 558 in the perturbed case. The mean running times were 0.03 seconds for the unperturbed case and 0.13 seconds for the perturbed case. In the unperturbed case, the support is recovered in 158 cases of 200 (79%) and 161 out of 600 in the perturbed case (26.8%). FISTA shows a false positive in 160 (80%) of unperturbed cases and of perturbed 534 (89%) cases. FISTA finds, in average, 0.56 false negatives in the unperturbed case and 2.38 false negatives in the perturbed case. FISTA also finds false positives in 64 out of 200 (64%) unperturbed cases and in 119 out of 600 perturbed cases (19.8%), finding in average 1 false positive per case for both perturbed and unperturbed cases.

The OMP algorithm uses a mean number of iterations that is equal to the number of atoms in the dictionary used for the representation. In the unperturbed case the OMP finds the correct support in 149 cases out of 200 (74,5% of cases) and in 226 cases out of 600 for the perturbed cases (37,6% of cases). For both the perturbed case and the unperturbed case the running time is, in average, around 0.001 seconds. In average, OMP has 0.67 false negatives in the unperturbed cases and 1.68 false negatives in the perturbed cases.

The FP-ADM algorithm uses 1000 iterations in each case in order to obtain the representations (the algorithm stops only at the iteration limit). The mean running times are 0.58 seconds for the unperturbed case and 0.49 for the perturbed case. The algorithm recovers the true support in 164 of the 200 cases (82% of the cases) for the unperturbed case and in 268 of 600 of the perturbed cases (44.7%). FP-ADM has instances in which it does not recover the signal in the unperturbed case. In average, FP-ADM has 0.21 false negatives in the unperturbed case and 1.06 false negatives in the perturbed case.

It can be seen that, both the FP based algorithms perform better than the OMP and FISTA. SQFP proves to be better than the FP-ADM having a better mean error and mean recovery error.

## 2.4 Generalized Gaussian Noise Case

Problems with Generalized Gaussian noise lead, through maximum likelihood, to optimization involving the $p$-norm. This is usually convenient for $p \geq 1$. However, for $p < 1$ the problems become non-convex and so harder to solve.

The sparse representation problem associated with Generalized Gaussian Noise is equation (2.17)

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

The problem can be formulated for $p > 0$, but we confine our study to $p \geq 1$, when the $p$-norm is truly a norm.

In most cases, the shape parameter $p$ is assumed to be known. However, using a different $p$ can lead to larger approximation errors. Only if the shape parameter value is close to the true one, we can hope to get a solution $x$ that is closer to the true solution.

We propose Algorithm 2.2, which consists of a modification for the Feasibility Pump (FP).

The algorithm repeatedly solves two problems, in which the binary variable $b$ is relaxed to the interval $[0,1]^n$. The solution is then rounded to $\widetilde{b}$, the nearest binary vector with $K$ values of one. One of the problems is (2.17), which is reformulated as a convex optimization program

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

**Algorithm 2.2:** Modified Feasibility Pump

```
   Data: Signal to represent $y \in \mathbb{R}^m$, dictionary $D \in \mathbb{R}^{m \times n}$, sparsity level $K \in \mathbb{Z}$,
         maximum number of iterations $Iter$, weights $\alpha$, $\lambda$, $\gamma$
   Result: Sparse representation $x \in \mathbb{R}^n$
1  Solve relaxed (2.18) with $b \in [0,1]^n$. The vectors $x$ and $b$ are obtained.
2  Use rounding procedure to obtain vector $\widetilde{b}$.
3  while number of iterations $\leq Iter$ do
4      Solve problem (2.19) for $x$ and $b$.
5      if $b$ is integer then
6          exit loop;
7      end
8      Use rounding procedure to obtain vector $\widetilde{b}$
9      if cycle is detected then
10         Perturb $\widetilde{b}$
11     end
12     Update $\alpha \leftarrow \gamma\alpha$
13 end
14 Return $x$, optimized with a Linear Optimization Program with norm $p$ on the found
    support.
```

which represents the starting point of the Feasibility Pump and is solved at step 1.

The Feasibility Pump steps use the model

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

which is solved at step 4 of algorithm 2.2.

Models (2.18) and (2.19) are implemented using CVX [12] and are compared with the greedy OMP-$\ell_p$ algorithm [19] and the algorithm presented in [18], which is called LP_L1.

The numerical results are obtained using a testing scheme similar to that used in previous chapters, with the significant distinction that the perturbation noise is now Generalized Gaussian.

The Feasibility Pump algorithm adapted for the Generalized Gaussian noise, called FP-GGN, is compared with the OMP modification presented in [19] called OMP-p and with the algorithm presented in [18], which is called LP_L1.

The dictionaries and test signals are generated similarly as in the Laplacian case. The $\lambda$ value selected for FP-GGN is fixed at 0.008 and for LP_L1 is fixed at 0.0025.

In order to compute the representation error, a formula which uses the $l_p$ norm is used

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

where $x$ is the computed solution.

The relative recovery errors is computed using

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

which is a similar formula to (2.10), but the $l_p$ norm replaces $l_2$, where $x_{true}$ is the true representation of the signal $y$ using the dictionary $D$ for the unperturbed case.

Figures 2.5 show the mean representation error of FP-GGN, computed using (**??**) for all test cases with $K = 5$ and condition number 10. It can be seen that for all cases, the errors follow the noise amplitude for all values of $p$. The behavior is similar to the models used for the Laplacian noise case and for the Gaussian noise case.

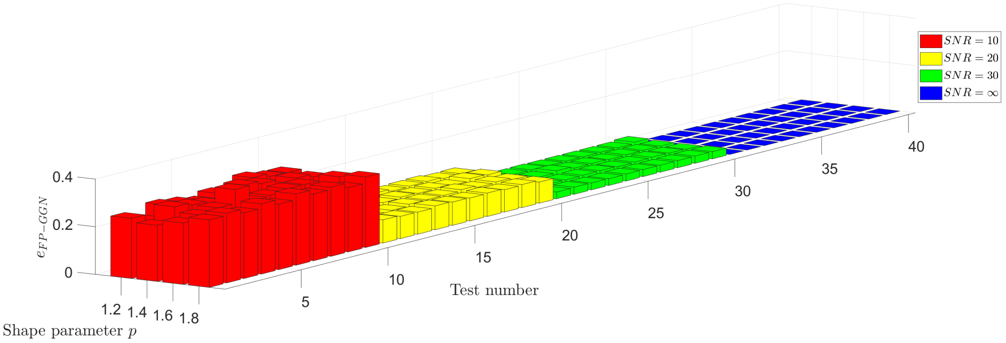

Figure 2.5: Mean representation errors for all the tested norms for FP-GGN for $K = 5$ and condition number 10

The mean representation errors are displayed in table 2.3, for the condition numbers 10 and for $K = 5$. It can be seen that only FP-GGN recovers the unperturbed signal, while LP_L1 seems to have the most difficulty in this case. In the perturbed case, OMP-p has the worst recovery errors in most cases especially for small SNRs. In most cases FP-GGN offers the best mean recovery error, but LP_L1 has cases in which it has the better mean recovery error especially for large SNRs.

Table 2.3: Mean Error for OMP-p, FP-GGN and LP_L1 for condition number 10 and $K = 5$

| $p$ | $SNR$ | 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 | $\mathbf{2.841 \cdot 10^{-16}}$ |

In the cases in which $p = 1.8$, FP-GGN needs, in average, 3.9 seconds (3 iterations) for the unperturbed case and 5.3 seconds (3.7 iterations) for the perturbed case. LP_L1 has a mean runtime of 0.44 seconds (53 iterations) for the unperturbed case and 0.70 seconds (224 iterations) for the perturbed case. OMP-p uses 0.51 seconds per unperturbed case and 0.74 seconds per perturbed case.

FP-GGN recovers the true support in all unperturbed cases and in 265 of the 600 perturbed cases (44.1%). LP_L1 recovers the true support in 170 out of 200 unperturbed cases (85%) and 204 out of 600 perturbed cases (34%). OMP-p recovers the support in 164 out of 200 unperturbed cases (82%) and in 255 out of 600 (42.5%) perturbed cases.

FP-GGN has, in average, 1.14 false negative per perturbed case, while LP_L1 has 0.16 false negatives per unperturbed case and 1.67 false negatives per unperturbed case. OMP-p has 0.5 false negatives per unperturbed case and 1.30 false negatives per pertrubed case.

In most tests, FP-GGN has the smallest mean recovery of all three algorithms and it also recovers the signal much better than OMP-p and LP_L1, having the least number of false negatives.

# 3 Shape parameter estimation

## 3.1 General shaper parameter estimation framework

In general, a sparse representation algorithm (SRA) has underlying assumptions of the noise characteristics.

Assuming that we have noise only within the GG family, with unknown shape parameter $p_{\text{true}} \geq 1$, there are also SRAs that can work with any given $p$, like OMP-p and LP_L1. The problem is that the shape parameter $p_{\text{true}}$ is often not known a priori. So, in most of the cases, if we compute the error $y - Dx$ obtained by a FSRA and then apply the shape parameter estimation (SPE) algorithm from [8], the resulting shape parameter $\tilde{p}$ is clearly different from $p_{\text{true}}$ as well as from the $p$ we have used.

Our purpose is the estimation of $p_{\text{true}}$ using a FSRA (Flexibile SRA). We propose two variants of an algorithm which computes an estimation of the shape parameter, together with the sparse representation.

The idea for both versions is to start with a given $p$ (we start from 2, as in most cases the noise is Gaussian). At each step of our iterative algorithm, the error $y - Dx$ is computed and the SPE algorithm [8] is used for this error in order to compute the associated $\tilde{p}$.

The difference between the two versions of the algorithm is given by the updates of the shape parameter $p$. For the first variant of the algorithm, named Half-Adapt, the norm is updated via

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

So, we go towards $p_{\text{true}}$ as guided by the empirical noise distribution, but temper the change in $p$ in order to prevent oscillations. The sparse representation is computed with the new $p$ and so on.

The second variant of the algorithm, named Mean-Norm, the norm is updated via

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

For this second variant of the algorithm, at the end an additional step is added where the $p$ is computed as the mean of all values of $p$ that were found during each iteration using the algorithm from [8], except for the $p = 2$ that is set at the initial step. After the $p$ is computed, the FSRA is ran one more time with the mean norm found in order to get the error.

The general algorithm used to estimate SPE is represented by 3.1.

**Algorithm 3.1:** General form for the algorithm for shape parameter estimation

```
Data: Signal to represent $y \in \mathbb{R}^{m \times t}$, dictionary $D \in \mathbb{R}^{m \times n}$, sparsity level $K \in \mathbb{Z}$,
      maximum number of iterations for $\ell_p$ norm estimation $Iter_{norm}$, stopping
      threshold $\theta$
Result: Sparse representations $x \in \mathbb{R}^{n \times t}$, estimated shape parameter $p \in \mathbb{R}$, $p \geq 1$
1 Compute sparse representations $x$ using algorithm with $p = 2$ for each column of $y$
2 Use algorithm in [8] to estimate shape parameter $\tilde{p}$ from representation errors
3 Update norm $p$ using (3.1) or (3.2)
4 while number of iterations $\leq Iter_{norm}$ or $|p - \tilde{p}| > \theta$ do
5     Compute sparse representations $x$ using algorithm with $p$ from the previous step
6     Use algorithm in [8] to estimate shape parameter $\tilde{p}$
7     Update $p$ using (3.1), Half-Adapt, or (3.2), Mean-Norm
8 end
9 If the Mean-Norm variant is used, the number of iterations was larger than 1 compute
    the norm $p$ as the mean value of all the previous values of $\tilde{p}$. Compute sparse
    representations $x$ using algorithm with the obtained $p$.
```

These frameworks are designed so that any FSRA can be used with it. The algorithm starts by running the FSRA, at step 1, for norm $l_2$. Using the SRE algorithm, $\tilde{p}$ is computed at step 2. The new $p$ is computed using $\tilde{p}$ with update (3.1) or (3.2) at step 3. At each iteration, the FSRA is evaluated, at step 4, with the new value of $p$ for the SPE and the cycle is repeated. The algorithm stops after the difference between $|p - \tilde{p}|$ is under a certain threshold $\theta$. Since it is unreasonable to aim for a very precise estimation of $p_{\text{true}}$, we keep $\theta$ large enough. Also, we limit the number of steps to $Iter_{norm}$, in order to stop possibly erratic behavior. After the algorithm stops, at step 9, if the update (3.2) is used, after the main loop is completed, the mean of all obtained values of $p$ is computed and the FSRA is evaluated for the obtained value. This is the final value for $p$.

Both adaptations are tested for the Generalized Gaussian Feasibility Pump, the OMP-$\ell_p$ algorithm [19] and the LP_L1 algorithm [18]. They are also compared with the versions of algorithms that know the true value $p$. It is shown that both versions can find shape parameters that are close to value of $p_{true}$. Also, the mean representation and mean recovery error are similar to the algorithms which use the real value of $p$. Also, it can be seen that Mean-Norm is faster than Half-Adapt and it also offers better support recover.

The testing setup is similar to the one used for the Generalized Gaussian, with dictionaries of size 50 × 100 being generated and using the same values for $K$, $SNR$, $\lambda$ and $p$. For each combination $K$, $p$, we test 5 different data sets. The algorithms that are integrated in the two variants will be FP-GGN, OMP-p and LP_L1.

The algorithms are compared in terms of mean representation errors, recovery errors and estimated shape parameters. The relative representation error is computed using (2.20) and the recovery error is computed using (2.21).

For the first case, the SNR for the perturbation signal is set at 30. The shape parameter estimation is analyzed over 5 runs. For all three algorithms the two frameworks are tested. In table 3.1, the value of the norm, for each pair of cells Half-Adapt and Mean-Norm, for their respective value of $K$ and $p$, that is closest to the real value of $p$, is written in bold. It can be seen that Mean-Norm offers the closest approximation to the real value of $p$ in most cases. For $K = 5$ and $K = 7$ FP-GGN offers the closest approximation. As the values increase LP_L1 appears to have the closest approximation.

Table 3.1: Mean Estimated Shape Parameters for OMP-p (top in each cell), FP-GGN (middle) and LP_L1 (bottom)

| $K$ | 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** |

In table 3.2, the mean representation errors obtained during the run of the frameworks are displayed. For each $K$, there are three columns: the left one contains the errors for Half-Adapt framework; the center one contains the errors for the Mean-Norm framework, the right one contains the errors of the algorithms that are in possession of $p_{true}$ for each value of $K$ and $p$, the smallest recovery error is written in bold.

The representation error increases as the value of $K$ increases. FP-GGN and LP_L1 have similar values for the representation errors, with very small differences. As $K$ increases, FP-GGN begins to have a slightly smaller representation error when compared to LP_L1, with OMP-p having the biggest error.

Table 3.2: Mean Representation Errors for OMP-p (top in each cell), FP-GGN (middle) and LP_L1 (bottom)

| $K$ | 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 |

Figures 3.1 contains the full information for all runs with $K = 5$ for the Mean-Norm framework, while figures 3.2 contain the full information for all runs with $K = 5$ for the Half-

Adapt framework. In these figures, the mean is being displayed with a different symbol; the horizontal displacement is used only for better visibility; the values $p_{\text{true}}$ are the same for all algorithms.

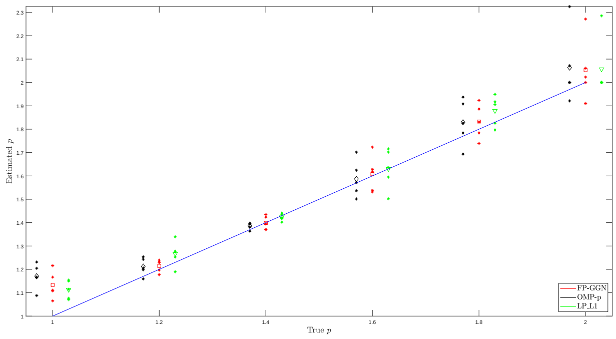

Figure 3.1: Shape parameter estimation for the noise for the $K = 5$ case using the Mean-Norm Algorithm

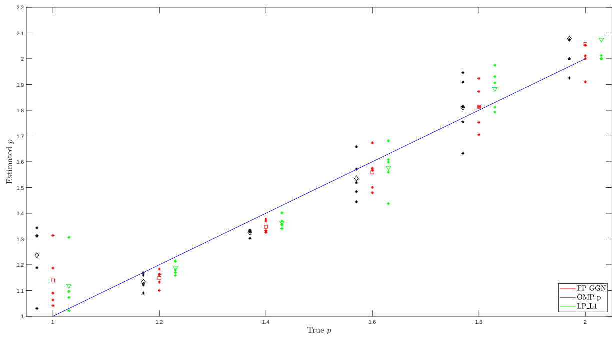

Figure 3.2: Shape parameter estimation for the noise for the $K = 5$ case using the Half-Adapt Algorithm

It can be seen, that as $K$ increases the shape parameter estimation tends to worsen. For the Mean-Norm framework, all algorithms begin to underestimate the value of $K$. For the case of the Half-Adapt framework, all algorithms, also, show a tendency of underestimating the value ok $K$. We note that the shape parameter is recovered well enough, especially for the lower values of the sparsity level $K$. The worst approximations appear when $p_{\text{true}} = 1$; a contributing cause is the fact that values $p < 1$ are not possible, hence a certain inherent bias.

The errors of the adaptive algorithms are near from those of the algorithms knowing $p_{true}$, occasionally better; so, our proposed framework is able to provide good representations even though $p_{\text{true}}$ is unknown. FP-GGN offers better error recovery in most cases. The Mean-Norm framework offers slightly better mean recovery error than the Half-Adapt framework and offers a better estimation for the shape parameter.

## 3.2 Application to images

We consider an image $Y \in \mathbb{R}^{t \times t}$ that is perturbed by a Generalized Gaussian signal $V \in \mathbb{R}^{t \times t}$ with zero mean and a standard deviation $\sigma = 1$, which is obtained by sampling a distribution with shape parameter $p$. We consider the case with additive noise, and the perturbed image $\widetilde{Y} \in \mathbb{R}^{t \times t}$ is obtained with

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

Using a small number of patches $P \in \mathbb{R}^{s \times s}$, with $s \leq t$ and $m = s \times s$, from the perturbed image $\widetilde{Y}$, a dictionary $D$ is trained. After the dictionary is obtained from the perturbed image, multiple patches $P$ are selected and a signal to be represented $Y^*$ is obtained. The optimization problem which results is

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

where $q$ is the number of patches used for the denoising problem.

Problem (3.4) can be reformulated as a LASSO problem, which gives us the optimization program

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

In practice, for each patch, the optimization problem (2.2) is solved. The image is then reconstructed, using the obtained patches $Y^*$.

The idea is, that after the learning process is completed, in the dictionary $D$ there are atoms which mostly represent the unperturbed image and atoms which mostly represent the noise. When the image is reconstructed using the obtained sparse dictionary $D$, the $K$ atoms selected will contain most of their information from the unperturbed image.

The image denoising framework, which can be derived, is shown in algorithm 3.2, where at each step the representation, each patch is computed using the trained dictionary $D$. Using the obtained representation of the patches $X$, the image is the reconstructed and a new image $Y_{rec}$ is obtained. The algorithm 3.1 will be used with the OMP-p algorithm.

Both variants of this algorithm, Half-Adapt and Mean-Norm are tested and compared against OMP and OMP-p with the known value of $p_{true}$. It can be seen that Half-Adapt has the best values for PSNR and the SSIM, when compared with all the algorithms. Mean-Norm is better than OMP, estimates better the shape parameter $p$ than Half-Adapt and is faster than Half-Adapt. It can be seen that the runtime and the quality of the estimation depends on the shape parameter update used.

The dictionary is trained using 2560 of patches from the perturbed image, using a dictionary learning procedure which uses OMP-p. Another larger set of 15876 patches is then extracted and is used for the actual process of image denoising.

The dictionary size selected is of 64 × 128. The patches are of size 8 × 8. The sparsity level is $K = 8$. The shape parameter are $p \in \{1.2, 1.4, 1.6, 1.8\}$. The values of SNR is chosen as 20. Tests are done on three images: Lena, Barbara and Boat. From each image, a large patch of 256 × 256 is selected from which the training and learning patches are extracted. Each image is perturbed with noise obtained by sampling a Generalized Gaussian noise with a shape parameter $p$. For each combination of image and value of $p$, 5 tests are done.

**Algorithm 3.2:** Algorithm for image denoising

```
Data: Set of patches to be represented $Y \in \mathbb{R}^{m \times q}$, dictionary $D \in \mathbb{R}^{m \times n}$, sparsity level
      $K \in \mathbb{Z}$, maximum number of iterations for $\ell_p$ norm estimation $Iter_{norm}$, stopping
      threshold $\theta$
Result: Sparse representations $X \in \mathbb{R}^{n \times q}$, estimated shape parameter $p \in \mathbb{R}$, $p \geq 1$
Train the dictionary using any algorithm which optimizes using the $p$ norm
Compute set $X$ using algorithm with $p = 2$ for each column of $Y$
Use algorithm in [8] to estimate shape parameter $\tilde{p}$ from representation errors
Update norm $p$ using (3.1) or (3.2)
while number of iterations $\leq Iter_{norm}$ or $|p - \tilde{p}| > \theta$ do
    Compute sparse representation set $X$ for each column of $Y$ using algorithm 3.1 with
      $p$ from the previous step
    Use algorithm in [8] to estimate shape parameter $\tilde{p}$
    Update norm $p$ using (3.1), for the Half-Adapt variant, or (3.2), for the Mean-Norm
      variant
end
If the Mean-Norm variant is used and the number of iterations was larger than 1,
  compute the norm $p$ as the mean value of all the previous values of $\tilde{p}$.
Compute sparse representation set $X$ using algorithm with the obtained $p$.
The resulting patches are used to reconstruct the image by computing for each pixel of
  the image the mean for all the patches which contain that pixel.
```

In figure 3.3, the original image (on the left), the perturbed image (center) and the denoised image (on the right) obtained using the Half-Adapt version of the algorithm, are presented, in order to show how the image is modified, when it is processed using the framework 3.2. It is obvious that the image quality is improved and most of the noise is removed from the perturbed image.

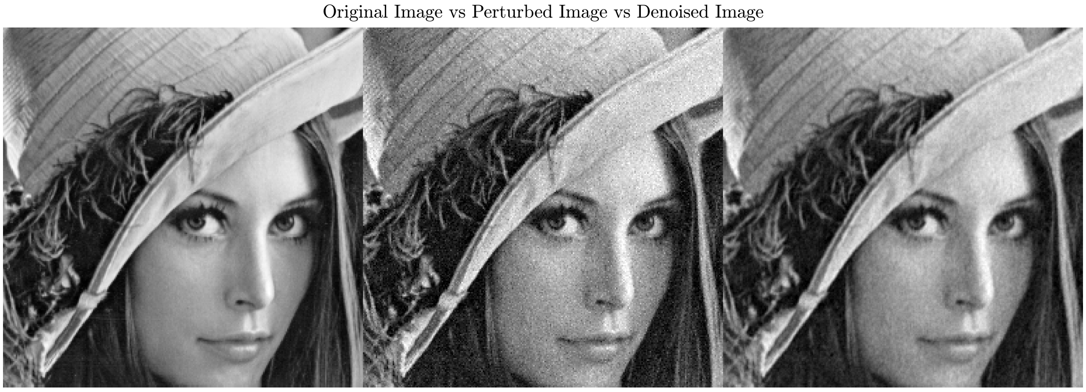

Figure 3.3: Image denoising for $K = 1.2$, for the Half-Adapt algorithm for the Lena test image

Table 3.3 shows the mean shape parameter estimation for the Mean-Norm and Half-Adapt version of the algorithm 3.2. In bold, the estimation that is closest to the real value is written. It can be seen that the Half-Adapt version tends to underestimate the shape parameter, while the Mean-Norm version tends to overestimate. For $p = 1.2$, the Half-Adapt estimation of the parameter is closest to the real value of $p$, while for $p = 1.4$ and $p = 1.6$ Mean-Norm estimates closer to the real value, in many tests the value being estimated at 2. For $p = 1.8$, both versions estimate the shape parameter at 2.

Table 3.3: Mean Estimated Shape Parameters for Mean-Norm and 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** |

In table 3.4, the mean PSNR values are presented for OMP, OMP-p, Mean-Norm and Half-Adapt algorithms. For each value of $p$, the largest PSNR values are written in bold. Mean-Norm has a better PSNR than OMP, while having a smaller value than OMP-p. Half-Adapt has the best value in all cases. For $p = 1.8$ Mean-Norm and Half-Adapt have the same value as OMP, as the shape parameter is estimated with the value of 2.

Table 3.4: Mean PSNR for 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** |

The mean values of the SSIM are presented in table 3.5, with the best value written in bold. The behavior is like for the PSNR values. Mean-Norm is better than the OMP, but has a smaller value than the other algorithms. Half-Adapt proves to have once again the best result. For $p = 1.8$, all algorithms have the same result, as both variants of the algorithm 3.2 estimate the shape parameter to 2.

Table 3.5: Mean Structural Similarity Index for OMP, OMP-p, Mean-Norm and Half-Adapt

| | 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** |

In average, the running time for 15876 patches, for Half-Adapt is 5097 seconds, while for Mean-Norm the running time is 3779 seconds, for OMP-p with $p_{true}$ the running time is 751 seconds and for OMP the running time is 0.002 seconds. It can be seen that Mean-Norm is faster than Half-Adapt. The running time increases as the decrease of the value of the estimated shape parameter is attenuated.

For all cases, it can be seen that Half-Adapt gives the best value of the PSNR and the 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 $\ell_p$-$\ell_1$ 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 $\ell_p$-Space for Sparse Approximation”. In: *IEEE Trans. Signal Processing* 64.1 (2016), pp. 60–75.