# Inverse Problems Project Report

Pedro MACHADO SANTOS ROHDE
Rodrigo ZAMBRANA PRADO

January 2020

## Brief introduction

In this project, our main goal was to solve an inverse problem of the form $y = Hx + n$, with a sparse signal $x$, Gaussian noise $n$ and convolution matrix $H$. To do so we will make use of simulation following probability laws and descent algorithms for convex optimization, which will be introduced in parts I and II before being used to solve the inverse problem in part III.

## I Simulation of a Gaussian law truncated to positive values

Our first task is to simulate the truncated Normal law, whose probability density function is

$$f(x) = \frac{1}{K(m,\sigma^2)} \exp\left(-\frac{(x-m)^2}{2\sigma^2}\right) \mathbf{1}_{\mathbb{R}^+}(x) \text{ with } K(m,\sigma^2) = \sqrt{\frac{\pi\sigma^2}{2}}\left[1 + \operatorname{erf}\left(\frac{m}{\sqrt{2\sigma^2}}\right)\right] \tag{I.1}$$

and whose cumulative distributive function is given by

$$F(t) = P[X < t] = \frac{\operatorname{erf}\left(\frac{t-m}{\sqrt{2\sigma^2}}\right) + \operatorname{erf}\left(\frac{m}{\sqrt{2\sigma^2}}\right)}{1 + \operatorname{erf}\left(\frac{m}{\sqrt{2\sigma^2}}\right)} \tag{I.2}$$

with the function $\operatorname{erf}(x)$ defined as $\operatorname{erf}(x) = \frac{2}{\pi}\int_0^x \exp\left(-t^2\right)dt$.

### I.1 Cumulative distributive function inversion method

The first method we shall see for generating elements following (I.1) is the CDF inversion method. We know that $F(t) \in [0,1], \forall t$. By generating some $u \sim U([0,1])$ and plugging them into $F^{-1}(u)$, we obtain samples following the law defined by $f(x)$. In the truncated Normal distribution case, inverting the CDF given by (I.2) yields

$$F^{-1}(u) = m + \operatorname{erf}^{-1}\left[u + \operatorname{erf}\left(\frac{m}{\sqrt{2\sigma^2}}\right)(u-1)\right]\sqrt{2\sigma^2} \tag{I.3}$$

Generating $N = 100$ samples following this method gives us the histogram in figure I.1, to which we superpose the probability density function that we were trying to simulate (given at (I.1)).

### I.2 Accept-reject methods

Our second method for generating samples following (I.1) is the accept-reject algorithm. We choose a candidate law $q(x)$ such that $f(x) \leq Mq(x), \forall x \in \mathbb{R}$, where $1/M$ is the acceptance ratio and $M > 1$. We generate a sample $Y$ following the candidate law $q(x)$ and also a $u \sim U([0,1])$. If $u < \frac{f(Y)}{Mq(Y)}$, we accept $Y$ as a sample that follows $f(x)$, otherwise we repeat the process. To maximize the acceptance ratio, we shall choose $M$ as small as possible, that is, $M = \sup_{x>0}\left\{\frac{f(x)}{q(x)}\right\}$.

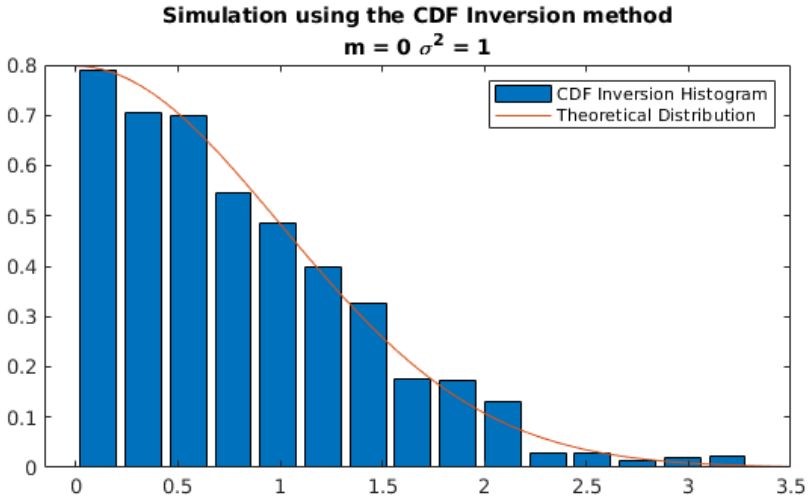


Figure I.1: Histogram comparison with the theoretical distribution using the CDF inversion method

### I.2.1 Normal candidate law

We choose the Normal law $\mathcal{N}(m, \sigma^2)$ as our first candidate law, which has the probability density function defined as

$$q(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-m)^2}{2\sigma^2}\right) \tag{I.4}$$

We find, from (I.1) and (I.4),

$$M = \frac{\sqrt{2\pi\sigma^2}}{K(m,\sigma^2)} = \frac{2}{1 + \operatorname{erf}\left(\frac{m}{\sqrt{2\sigma^2}}\right)} \tag{I.5}$$

We then simulate the truncated Normal law using this method to find the results in figure I.2.

### I.2.2 Exponential candidate law

Our next candidate law is an exponential distribution given by

$$q(x) = \lambda \exp(-\lambda x)\mathbf{1}_{\mathbb{R}^+}(x) \text{ with } \lambda = {}^{\left(\sqrt{m^2+4\sigma^2}-m\right)}/_{2\sigma^2} \tag{I.6}$$

We compute the constant $M$ that maximizes the acceptance ratio, finding

$$M = \frac{1}{\lambda K(m,\sigma^2)} \exp\left(\frac{\lambda^2\sigma^2 + 2m\lambda}{2}\right) \tag{I.7}$$

We then simulate the truncated Normal law using this method to find the results in figure I.3.

## I.3 Comparison of the methods

All three methods generated samples following the truncated Normal law, as seen in the comparisons of the histograms with I.1.

For the best performance of the accept-reject method, we should choose the candidate law with the highest acceptance ratio (i.e. ${}^1/_M$) for given parameters $m$ and $\sigma^2$. As the mean $m$ gets bigger and bigger, the truncated Normal law becomes ever more like the Normal law. Our $M$ will then approach 1

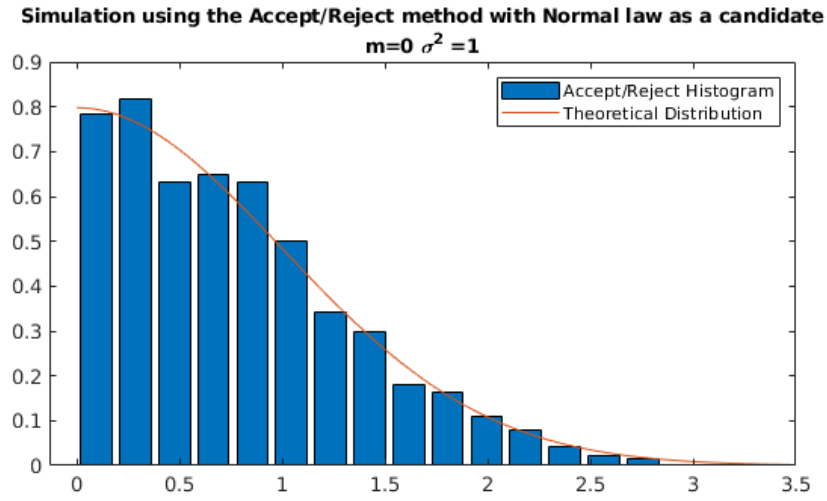


Figure I.2: Histogram comparison with the theoretical distribution using the accept/reject method with the Normal distribution as candidate

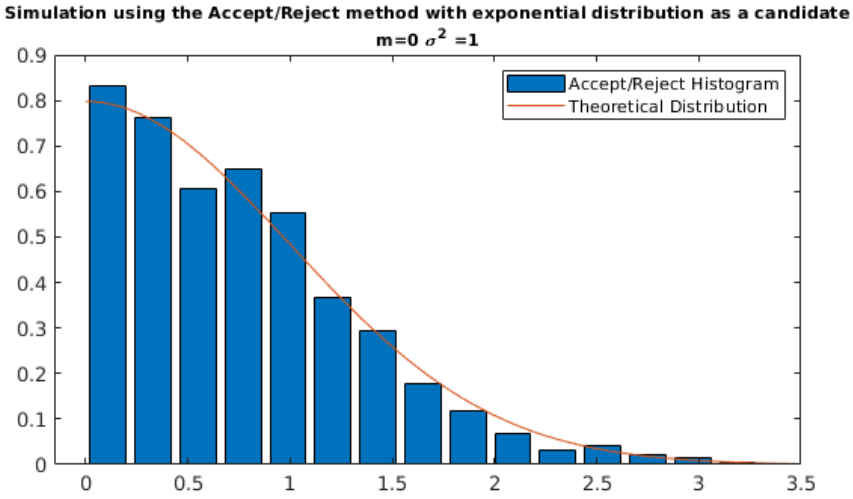


Figure I.3: Histogram comparison with the theoretical distribution using the accept/reject method with the Exponential distribution as candidate

and we will have an acceptance ratio of $1/M = 1$. For small values of $m$, it is the exponential distribution that best resembles the truncated Normal law. We can find a comparison between the two in figure I.4, where we plot $1/M$ for varying $\sigma^2$ and $m$. This theoretical result is then validated by figure I.5, where we plot the actual acceptance ratio measured experimentally. For any given mean and variance, one should then choose the candidate law which has the higher acceptance ratio i.e. higher $1/M$ i.e. smaller $M$.

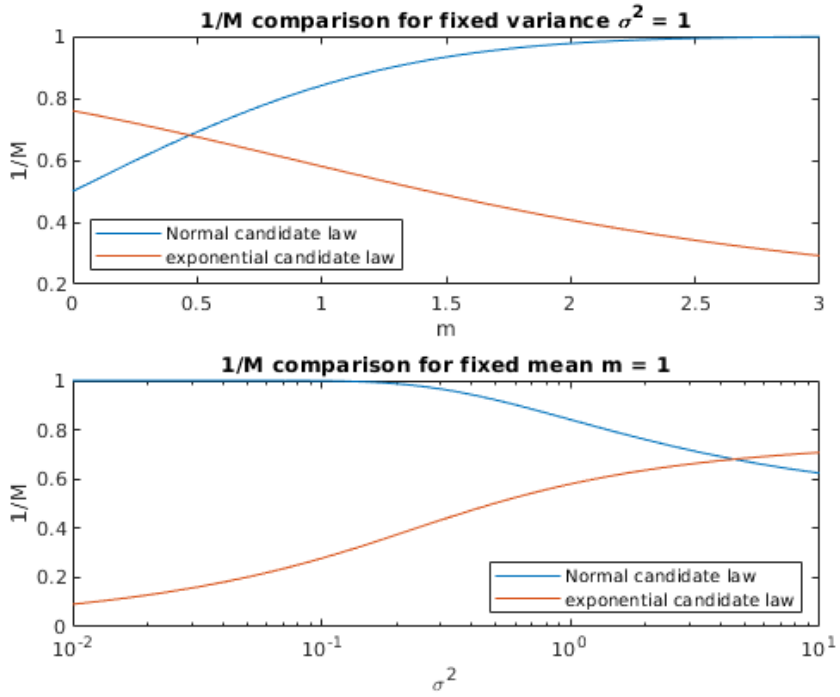


Figure I.4: Theoretical acceptance ratio comparison for varying $\sigma^2$ and $m$

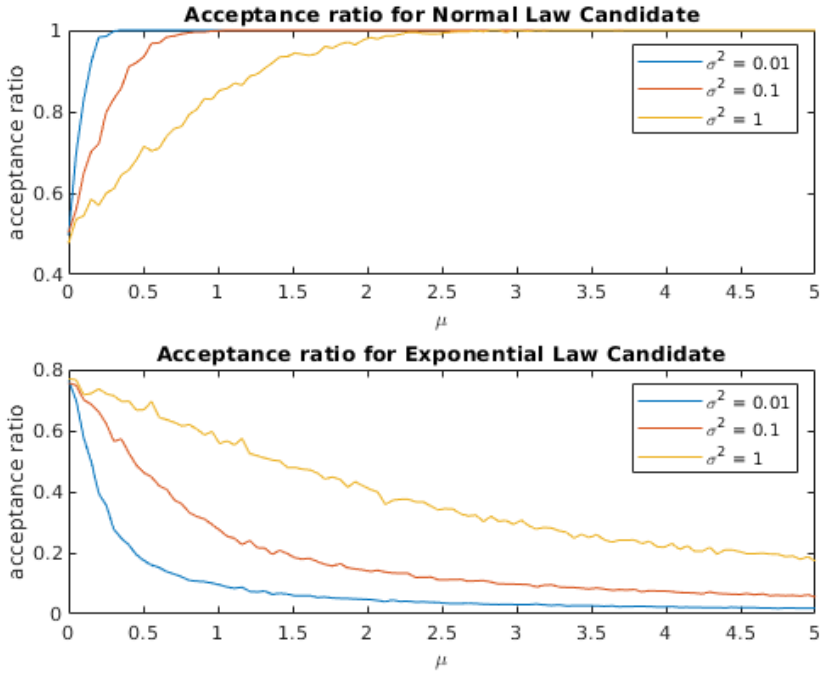


Figure I.5: Experimental acceptance ratio comparison for varying $\sigma^2$ and $m$

# II Descent Methods

The second task is to minimize the function given by

$$f_0(x) = a \arctan\left(dx_1^2 + x_2^2\right) + b\left(x_1 - x_1^{(0)}\right)^2 + c\left(x_2 - x_2^{(0)}\right)^2 \tag{II.1}$$

with $x = (x_1, x_2)^T$ and parameters $(a, b, c, d) = (0.5, 8, 0.55, 2)$ and $(x_1^{(0)}, x_2^{(0)})^T = (0.5, 1)^T$ chosen such that $f_0$ is convex.

We will look at two descent methods to minimize this function: gradient descent, a first order method, and the Newton method, a second order method. We will then compare them for different choices of parameters.

## II.1 Gradient descent

The principle of gradient descent is to find the direction of greatest descent of the function at the current point and execute a step in that direction. This direction is given by the negative gradient $-\nabla f_0(x^k)$ at the step $k$. We follow the update rule $x^{k+1} = x^k - \alpha \nabla f_0(x^k)$, where $\alpha$ is the step size that we will determine in two different ways: by having a fixed step size for all iterations ($\alpha^k = \alpha$) or by choosing $\alpha \in [0.001, 10]$ that maximizes the descent, that is, $\alpha^k = \arg\min_{\alpha \in [0.001,10]} f_0(x^k - \alpha \nabla f_0(x^k))$.

We compute the gradient of $f_0(x)$ given by (II.1), which yields

$$\nabla f_0(x) = \begin{pmatrix} b(2x_1 - 2x_1^{(0)}) + {}^{2adx_1}/_{[(dx_1^2+x_2^2)^2+1]} \\ c(2x_2 - 2x_2^{(0)}) + {}^{2ax_2}/_{[(dx_1^2+x_2^2)^2+1]} \end{pmatrix} \tag{II.2}$$

## II.2 Newton method

The Newton method is very similar to gradient descent, but our descent direction is calculated differently, using the Hessian of the function $\nabla^2 f_0(x)$, which makes this a second order descent method. Our update rule is now $x^{k+1} = x^k - \alpha \left(\nabla^2 f_0(x^k)\right)^{-1} \nabla f_0(x^k)$, where, just as before, $\alpha$ is the step size that we will either fix for every iteration or choose it such that it maximizes the descent at each iteration.

We find the Hessian matrix of $f_0(x)$ to be

$$\nabla^2 f_0(x) = \begin{pmatrix} 2b + \frac{2ad}{(dx_1^2+x_2^2)^2+1} - \frac{8ad^2x_1^2(dx_1^2+x_2^2)^2}{[(dx_1^2+x_2^2)+1]^2} & \frac{-8adx_1x_2(dx_1^2+x_2^2)}{((dx_1^2+x_2^2)^2+1)^2} \\ \frac{-8adx_1x_2(dx_1^2+x_2^2)}{((dx_1^2+x_2^2)^2+1)^2} & 2c + \frac{2a}{(dx_1^2+x_2^2)^2+1} - \frac{8ax_2^2(dx_1^2+x_2^2)}{[(dx_1^2+x_2^2)^2+1]^2} \end{pmatrix} \tag{II.3}$$

## II.3 Comparison of the methods

We can now compare both methods for different parameters. Table II.1 summarizes our experiments. Here, we compare the value for the function at the final point found by the algorithms, the number of steps it took to get there, and the time it took to run each algorithm in the same machine for $\epsilon_1 = 10^{-3}$ and $\epsilon_2 = 10^{-6}$.

All methods have converged to points that are relatively close to each other. We remark that, in the iterations it takes to go from $\epsilon_1$ to $\epsilon_2$, the descent gets much slower, due to the fact that the gradient is very small in this region.

In terms of number of iterations, the Newton method converges significantly faster. So fast that, even if it takes more calculations, it actually took less time to converge than the gradient descent methods. An optimal step size also represents an improvement in the number of iterations, but since it tests many (we chose 1000) update steps, it can become slower in terms of time.

We should also note that the times indicated in the table are dependent on the processing power allocated to the task by the computer. We tried to make all simulations under the same conditions, but some may have been influenced by other tasks being run by the computer.

| descent method | step size | $\epsilon$ | $x$ | $f_0(x)$ | iterations | time [s] |
|---|---|---|---|---|---|---|
| gradient descent | fixed $\alpha = 0.01$ | $\epsilon_1$ | $(0.4574, 0.4301)^T$ | 0.4646 | 165 | 0.0088 |
| | | $\epsilon_2$ | $(0.4664, 0.6505)^T$ | 0.4308 | 436 | 0.0090 |
| | fixed $\alpha = 0.1$ | $\epsilon_1$ | $(0.4636, 0.5893)^T$ | 0.4337 | 25 | 0.0089 |
| | | $\epsilon_2$ | $(0.4667, 0.6566)^T$ | 0.4308 | 52 | 0.0093 |
| | optimal $\alpha \in [0.001, 10]$ | $\epsilon_1$ | $(0.4662, 0.6402)^T$ | 0.4310 | 10 | 0.0131 |
| | | $\epsilon_2$ | $(0.4668, 0.6582)^T$ | 0.4308 | 16 | 0.0112 |
| Newton method | fixed $\alpha = 1$ | $\epsilon_1$ | $(0.4668, 0.6590)^T$ | 0.4308 | 5 | 0.0035 |
| | | $\epsilon_2$ | $(0.4668, 0.6590)^T$ | 0.4308 | 5 | 0.0036 |
| | fixed $\alpha = 10$ | $\epsilon_1$ | $(0.4476, 0.6099)^T$ | 0.4346 | 42 | 0.0058 |
| | | $\epsilon_2$ | $(0.4662, 0.6574)^T$ | 0.4308 | 75 | 0.0060 |
| | optimal $\alpha \in [0.001, 10]$ | $\epsilon_1$ | $(0.4667, 0.6582)^T$ | 0.4308 | 5 | 0.0049 |
| | | $\epsilon_2$ | $(0.4668, 0.6590)^T$ | 0.4308 | 6 | 0.0042 |

Table II.1: Descent methods comparison

# III Deconvolution of sparse signals

Now, for the final task, we are to solve the inverse problem given by $y = Hx + n$, where $n$ is a Gaussian noise $n \sim (0, \sigma^2 I)$ ($\sigma^2$ known), $H$ is a known convolution matrix and $x \in \mathbb{R}_+^N$ is a signal known to be sparse and positive ($x_i \geq 0$). We can see an example of what $x$ and $y$ look like in figure III.1. We will estimate $x$ from observations $y$ through the means of convex optimization similar to that of part II then also using the Gibbs sampling algorithm which will make use of the simulation of probability laws seen in part I.

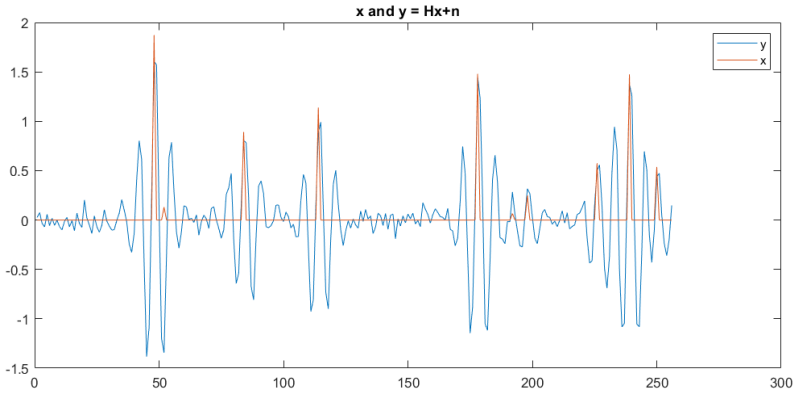


Figure III.1: Sparse (10 non-zero values) signal $x$ and $y = Hx + n$

## III.1 Convex optimization algorithm

Knowing $x$ to be sparse, our goal is to minimize the squared error between our observations $y$ and $Hx$ while also forcing $x$ to having small $\ell_1$-norm. We also know $x$ to be positive, so our problem becomes

$$\min_x \|y - Hx\|_2^2 + \lambda \|x\|_1 \text{ such that } x_i \geq 0, \forall i \tag{III.1}$$

We can separate this function $f_0(x)$ that we wish to minimize in a convex and differentiable term $f(x) = \|y - Hx\|_2^2$ and a convex but non-smooth function $g(x) = \|x\|_1$. The minimization of such $f_0(x)$ functions can be done by using the proximal descent method.

To understand this method, we first have to introduce the proximal operator, defined as $\text{prox}_{\lambda f}(x) = \arg\min_y \left(f(y) + \frac{1}{2\lambda}\|y-x\|_2^2\right)$.

The proximal gradient method for the function $f_0(x) = f(x)+g(x)$ is then very similar to the regular gradient method: its update rule is $x^{k+1} = \text{prox}_{\lambda g}(x)\left(x^k - \alpha^k \nabla f(x^k)\right)$, which is the proximal operator applied to a gradient descent step. When $g(x) = \|x\|_1$ (our case here), the proximal operator makes the update rule be

$$x_i^{k+1} = \begin{cases} \tilde{x_i} - \lambda, & \text{if } \tilde{x_i} \geq \lambda \\ 0, & \text{if } |\tilde{x_i}| < \lambda \\ \tilde{x_i} + \lambda, & \text{if } \tilde{x_i} \leq -\lambda \end{cases} \tag{III.2}$$

where $\tilde{x} = x^k - \alpha^k \nabla f(x^k)$ is the gradient descent step and $x_i$ denotes the $i$-th component of vector $x$.

Similarly to part II of this project, we choose a different step size $\alpha^k$ at each iteration, but here we use the following rule:

$$\begin{gathered} \text{initialize } \alpha^k = \alpha,\ 0 \ll \beta < 1 \\ \text{while } \|r^k\|_2^2 > \|r^{k-1}\|_2^2 - \nabla f(x^k)(x^{k-1} - x^k) + \frac{1}{2\alpha^k}\|x^{k-1} - x^k\|_2^2, \\ \text{do } \alpha^k = \alpha^k \beta \end{gathered} \tag{III.3}$$

where $r^k = y - Hx^k$ is the residual for $\alpha^k$.

We had said before that we know $x$ to be positive. To introduce this positivity constraint, we will, at each iteration, project $x^k$ onto $C = \{x; x_i \geq 0\}$. This is done simply by setting to zero all $x_i < 0$. All things considered (gradient descent, proximal operator, and projection onto $C$), our final update rule is

$$x_i^{k+1} = \begin{cases} \tilde{x_i} - \lambda, & \text{if } \tilde{x_i} \geq \lambda \\ 0, & \text{if } \tilde{x_i} < \lambda \end{cases} \text{ with } \tilde{x} = x^k - \alpha^k \nabla f(x^k) \tag{III.4}$$

Figure III.2 on section III.3 shows an example of deconvolution by the proximal gradient.

## III.2 Bayesian setting

We now wish to solve this problem in a Bayesian setting, by having a prior distribution for each element $x_i$ of $x$, which we assume to be independent. We know $x$ to be sparse, so we choose an exponential distribution for the $x_i$ priors, so as to have high probability of having it be a very small value:

$$f(x_i) = \lambda \exp(-\lambda x_i)\mathbf{1}_{\mathbb{R}^+}(x_i) \tag{III.5}$$

We will construct a Gibbs sampling algorithm to generate the $x_i$ accordingly. To do so, we must compute the posterior conditional law of each $x_i$ conditioned on the knowledge of $y$ and all the other elements of $x$, that is, $f(x_i|x_{j\neq i}, y)$.

We start by computing the posterior law of the vector $y$ conditioned on $x$. We know the noise to be zero-mean normally distributed, with each component of the noise vector $n_k \sim \mathcal{N}(0, \sigma^2)$. Since $y_k = H_{k,*}x$ ($H_{k,*}$ denotes the $k$-th line of $H$), this is equivalent to $y_k - H_{k,*}x \sim \mathcal{N}(0, \sigma^2)$, hence $y_k|x \sim \mathcal{N}(H_{k,*}x, \sigma^2)$, or

$$f(y_k|x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_k - H_{k,*}x)^2}{2\sigma^2}\right) \tag{III.6}$$

For all components of $y$, (III.6) becomes

$$f(y|x) = \prod_k f(y_k|x) = (2\pi\sigma^2)^{-\frac{N}{2}} \exp\left(-\sum_k \frac{(y_k - H_{k,*}x)^2}{2\sigma^2}\right) = (2\pi\sigma^2)^{-\frac{N}{2}} \exp\left(-\frac{\|y - Hx\|_2^2}{2\sigma^2}\right) \tag{III.7}$$

We now turn to $f(x)$. Since all $x_i$ are independent, this is just the product of all $f(x_i)$, hence

$$f(x) = \prod_i f(x_i) = \prod_i \lambda \exp(-\lambda x_i) \mathbf{1}_{\mathbb{R}^+}(x_i) = \lambda^N \exp\left(-\lambda \sum_i x_i\right) \mathbf{1}_{\mathbb{R}^+}(x) \tag{III.8}$$

We compute the posterior law of the vector $x$ conditioned on $y$, up to a constant factor:

$$f(x|y) \propto f(y|x) f(x) \propto \exp\left(-\frac{\|y - Hx\|_2^2}{2\sigma^2} - \lambda \sum_i x_i\right) \mathbf{1}_{\mathbb{R}^+}(x) \tag{III.9}$$

Finally, developing the expression and marginalizing over all $x_{j \neq i}$ yields the law we were looking for:

$$\begin{gathered} f(x_i | x_{j \neq i}, y) \propto \exp\left(-\frac{(x_i - m_i)^2}{2\sigma_i^2}\right) \mathbf{1}_{\mathbb{R}^+}(x_i), \\ \text{with } m_i = \frac{H_{*,i}^T \left(y - \sum_{j \neq i} x_j H_{*,j}\right) - 2\sigma^2 \lambda}{\|H_{*,i}\|_2^2} \text{ and } \sigma_i^2 = \frac{\sigma^2}{\|H_{*,i}\|_2^2}, \\ \text{where } H_{*,i} \text{ denotes the } i\text{-th column of } H. \end{gathered} \tag{III.10}$$

We recognize from (III.10) the truncated Normal distribution defined by (I.1), with mean $m_i$ and variance $\sigma_i^2$. We use the algorithms from part I - for simplicity we used the CDF inversion method (section I.1) - to construct a Gibbs sampler, which is a Markov chain Monte Carlo algorithm that generates the $x_i$ in the following way:

First we initialize $x$ as $y$ (this is just a matter of choice), that is, $x^0 = y$. Then for $K = 100$ iterations, at each iteration $k$ we generate each $x_i^k$ in order, following (III.10) and using the $x_{j \neq i}$ previously calculated by the algorithm, that is $x_1^k, \ldots, x_{i-1}^k, x_{i+1}^{k-1}, \ldots, x_N^{k-1}$. If ever at a certain iteration $k$ the mean $m_i^k$ calculated according to (III.10) is negative, we set $x_i^k$ at zero. Our final estimation for $x$ will then be $x^K$ from the last iteration.

In our experiments, we found a good value for $\lambda$ to be 100. This insures that the generated $x$ is sparse enough. Figure III.3 on section III.3 shows an example of deconvolution by this algorithm.

## III.3 Comparison of the methods

We now compare the proximal gradient method, the Gibbs sampler and the Orthogonal Matching Pursuit method. In our simulations, we generate 100 random spike trains with $N = 256$ samples and 10 spikes each and a SNR of 10dB for the noise. We then compare the average number of non-zero values and the mean squared error between $x$ and its approximation from observations $y$ found by the different methods. We also compare the algorithms in terms of their execution time. We simulate two types of $x$: with positive or real-valued $x_i$. When $x_i \geq 0$, we introduce the positivity constraint discussed before to the proximal gradient method, but not to the Orthogonal Matching Pursuit method. Table III.1 summarizes our results.

| method | x positive | relative MSE | average number of non-zero values | average time [s] |
|---|---|---|---|---|
| Orthogonal Matching Pursuit | no | 0.43336 | 15 | 0.0017 |
| | yes | 0.47622 | 16.74 | **0.0016** |
| Proximal Gradient | no | 0.1925 | 16.91 | 0.7666 |
| | yes | **0.12737** | 12.43 | 0.3961 |
| Gibbs Sampler | yes | 0.14304 | **12.15** | 2.6044 |

Table III.1: Deconvolution methods comparison

We find that the Orthogonal Matching Pursuit method is by far the fastest, but its results are far behind those of the proximal gradient and Gibbs sampler algorithms. Between those two we observe

very close results, with the MSE of the proximal gradient better and the number of non-zero components of the Gibbs sampler slightly better. The main disadvantage of the Gibbs algorithm is its execution time, with an average 2.6 seconds to estimate an $x$, way slower than the other two. This is due to the high number of iterations we make it run. Since it has no other stopping criterion (like the proximal gradient's $\epsilon$), it just runs the fixed $K = 100$ iterations at each time, generating $N = 256$ $x_i$ at each iteration. However, if we lower the number of iterations $K$, the performance of the algorithm decreases.

Figures III.2 and III.3 shows the reconstruction of the same $x$ as in III.1, estimated from the same $y = Hx + n$, with the proximal gradient and the Gibbs sampling algorithm methods, respectively. We remark that the positions of the highest spikes are well approximated, but the amplitude of the spikes and the position of some of the smaller ones could be better estimated.

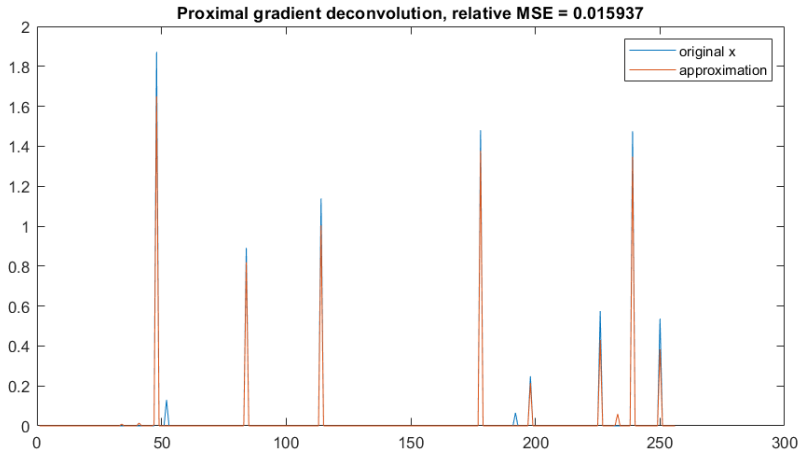


Figure III.2: Proximal gradient descent with positive $x$. $\hat{x}$ had 13 non-zero values (vs. 10 from $x$)

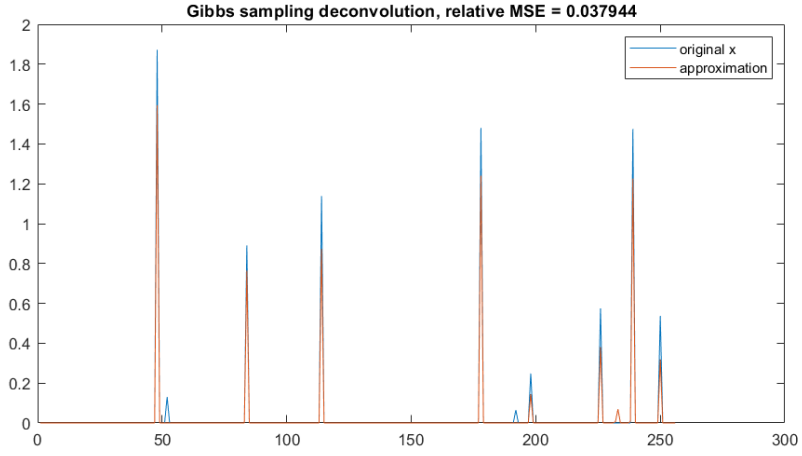


Figure III.3: Gibbs sampling algorithm to estimate $x$. $\hat{x}$ had 11 non-zero values (vs. 10 from $x$)

## Concluding remarks

In this project, we solved the inverse problem $y = Hx + n$ with sparse $x$. To do so, we used a slightly modified version of the descent algorithms seen in part II and we also developed a Gibbs sampler that used the algorithms seen in part I to generate the components of $x$. The descent and simulation following a probability law algorithms presented here could be used to solve various other inverse problems in different forms. The results were consistent with what was expected from the theoretical calculations and we presented means of choosing which method is best suited to a given problem.