Linear estimation problem
Markov Chains and Algorithmic Applications

Henry Declety, Olivier Couque, Arnaud Duvieusart

***Abstract*— This paper presents our work on the Markov Chain Monte-Carlo (MCMC) method for a generalized linear estimation problem. In this project we tried different algorithms in order to retrieve a vector after a linear transformation**

I. Description

The aim of the project is to retrieve a vector $X = (X_1, ..., X_n)^T \in \{-1, 1\}^n$ from the following m observations :

$$Y_i = \phi(\sum_{j=1}^{n} \frac{W_{ij} X_j}{\sqrt{n}})$$

Where W is a $m \times n$ known normally distributed matrix, and $\phi : x \to max(0, x)$ We will try to approch X by $\hat{x}$ by minimizing the *enegry* denoted by

$$H_Y(\hat{x}) = \sum_{i=1}^{m} \left| Y_i - \phi(\frac{Z_i}{\sqrt{n}}) \right|^2$$

With Z_i the *ith* element of $W.\hat{x}$

II. Estimation

To estimate our vector X, we used the inverse temperature β as a parameter. This parameter will vary during the execution to tend towards infinity. The choice of the increasing rate is discussed later on. We implemented two algorithms for this problem:

A. Metropolis Chain algorithm

The metropolis chain algorithm works as follow : You initialize randomly the vector $\hat{x}^0$. At each step *t* you create the vector $\hat{x}^{t+1}$ by choosing randomly one of its components and flip its sign with probability

$$p = min(1, e^{-\beta(H_Y(\hat{x}^{(t+1)}) - H_Y(\hat{x}^{(t)}))})$$

B. Glauber algorithm

Similarly, we start with a random vector but at step *t* we reset the value of a randomly picked coordinate to ± 1 with probability

$$p = \frac{1 \pm x_i^{(t)} tanh(\beta(H_Y(\widetilde{x}^{(t)}) - H_Y(x^{(t)})))}{2}$$

With $\widetilde{x}$ is the vesrion of x with the *ith* component flipped.

III. Parameters estimation

Though we want to minimize the energy of our estimation, we are also interested by the *error*. This error is the fraction of entries on which $\hat{x}$ differs from X and given by

$$e(\hat{x}, X) = \frac{1}{4n} \|\hat{x} - X\|^2$$

Proof :

$$e(\hat{x}, X) = \frac{1}{4n} \sum_{i=1}^{n} (\hat{x}_i - X_i)^2$$

$$= \frac{1}{4n} \sum_{i=1}^{n} 4 \times \mathbf{1}\{\hat{x}_i \neq X_i\}$$

$$= \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}\{\hat{x}_i \neq X_i\}$$

where the second line comes from the fact that $\hat{x}_i$ and X_i take values in $\{-1, 1\}$ for any i, thus $(\hat{x}_i - X_i)^2$ is 0 for equal components and is 4 otherwise.

A. Simmulated annealing

For this section, we ran all our experiments with parameters $m = 2000$ and $n = 1000$ such that the running times of the algorithms are not too long compared to the ones with higher dimensions matrices.

By running the Metropolis or Glauber algorithms with a random initial vector $\hat{x}$ for a sufficiently large number of iterations, the error term tends to 0 (the approximate vector $\hat{x}$ is equal to X). What is interesting is to try different strategies in order to find the convergence of error to 0 as fast as possible. To do so, we tried different settings:

1) increase β additively at some iterations (e.g. every 100 iterations)
2) increase β multiplicatively at some iterations
3) increase β additively at each iteration after a given iteration (e.g. iteration 1000)
4) increase β multiplicatively at each iteration after a given iteration
5) increase β additively when the difference of errors in a given window is below a given threshold.

For each setting and each algorithm, we performed a grid search to estimate the best parameters that give the minimal error at the $10'000^{th}$ iterations. These parameters are: the

starting value for β (all settings), the rate to increase β (all settings), the number of iterations before increasing β (setting 1 and 2), the best threshold from which we start increasing β (setting 3 and 4), the size of the window to compute the difference of errors (setting 5), the threshold value for which we increase β if the difference of errors is below this threshold (setting 5).

While performing the grid searchs, we ran 4 times each algorithm with the same parameters in order to find a mean for the minimal number of iterations. This is important as we know that the Glauber and Metropolis algorithms depend on randomness. We found that increasing β additively when the difference of errors for a given window is below a given threshold (setting 5) gave us the minimal losses compared to the other settings. In fact with this setting, β is increased in a "smart way" compared to the other settings. Thus we will only use this setting for the rest of the report.

After computing the grid search to find the best parameters for one of the algorithm (e.g. Metropolis), we kept the few set of parameters that gave an error below 0.01 and re ran the algorithm 10 times with 10'000 iterations for each set in order to have a better approximation of the mean of the final error (maybe we had luck during the grid search). We kept the parameters giving the lowest mean error for both algorithm.

For the Metropolis algorithm we found the best parameters to be: $\beta_0 = 3$, increase rate of 0.07, a size of window of 152 iterations and a threshold value of 0.00075. These parameters allow us to reach a mean error of 0.0093 at iteration 10'000. Figure 1 shows a part of the results of the grid search for the Metropolis algorithm with set parameters $\beta_0 = 3$ and window size of 152 iterations.

For the Glauber algorithm we found the best parameters to be: $\beta_0 = 2$, increase rate of 0.07, a size of window of 233 iterations and a threshold value of 0.001. These parameters allowed us to reach a mean $error = 0.026$ at the $10'000^{th}$ iteration. We obtain lowest errors with the Metropolis algorithm than with Glauber algorithm

We also wanted to compare the evolution of the average errors of both algorithms with the best computed parameters against a baseline in which the β would be fixed at the beginning and would not change over time (same initial β between best and baseline). Figure 2 shows this evolution for both Metropolis and Glauber algorithms. We can deduct that increasing β with our set of parameters allow reaching a low error faster than not increasing it.

Figure 3 shows the evolution of the errors and betas for one run of the Metropolis algorithm. We clearly see that when β starts increasing the error curve for the best Metropolis goes well below the error curve of the baseline.

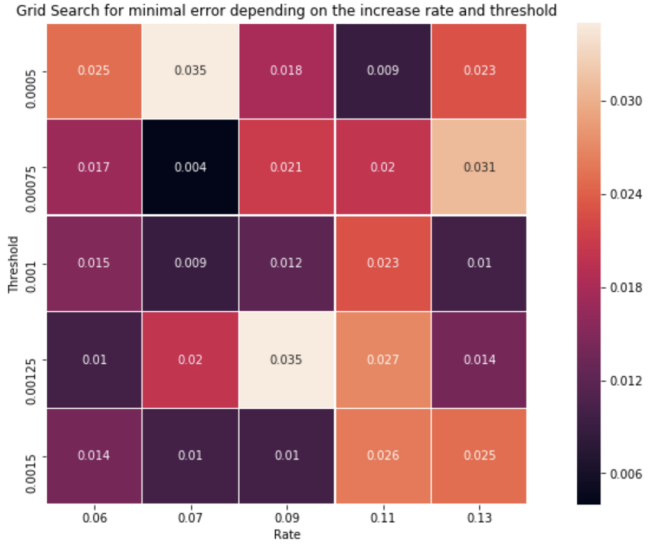

Fig. 1: Grid Search results for the Metropolis Algorithm with initial $\beta = 3$ and window size of 152 iterations: only the threshold value and the increase of beta vary. The numbers in the squares indicate the mean error value at iteration $10'000$

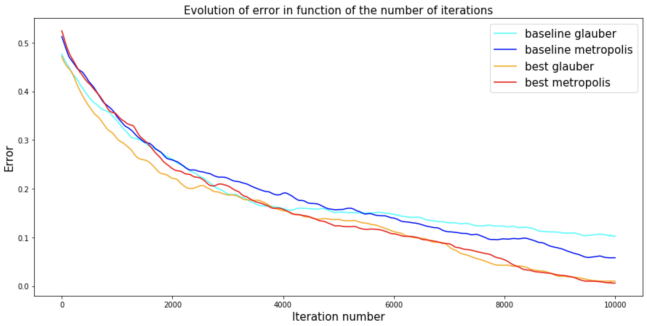

Fig. 2: Comparison of Metropolis and Glauber errors over time: baseline VS with best parameters

IV. Error and critical threshold

With our parameters determined, we analyze the expected error and the standart deviation of the error. As this error will vary along with the dimension of the Matrix W we plot these values as a function of $\frac{m}{n}$. The results are shown in Figure 4. This was obtained with 15 executions of the Metropolis Chain algorithm with 10'000 iterations for 50 different ratios of $\frac{m}{n}$. We observe that for a ratio close to 0, our algorithm is not better than a random guess. This makes sense as $\frac{m}{n}$ tends towards 0 implies that we have much more dimensions to guess than what we have to train our model. As this ratio increases, the expected error decreases linearly and reaches 0 when m is roughly twice as big as n. The standard deviation of the error has an interesting behavior. It remains globally constant until the expected error approches 0. It is at this point that the standard deviation is maximum. The explanation is the following : it is around this value that some of our experiments gave an error close to 0. But for other experiments we reach a local minimum

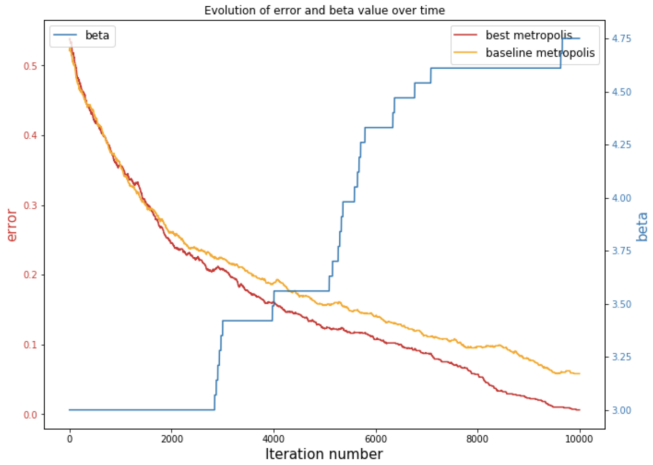

Fig. 3: Evolution of the errors and betas over time for the Metropolis algorithm: the β evolution corresponds to the curve of errors for Metropolis with best parameters (the β is constant with baseline Metropolis)

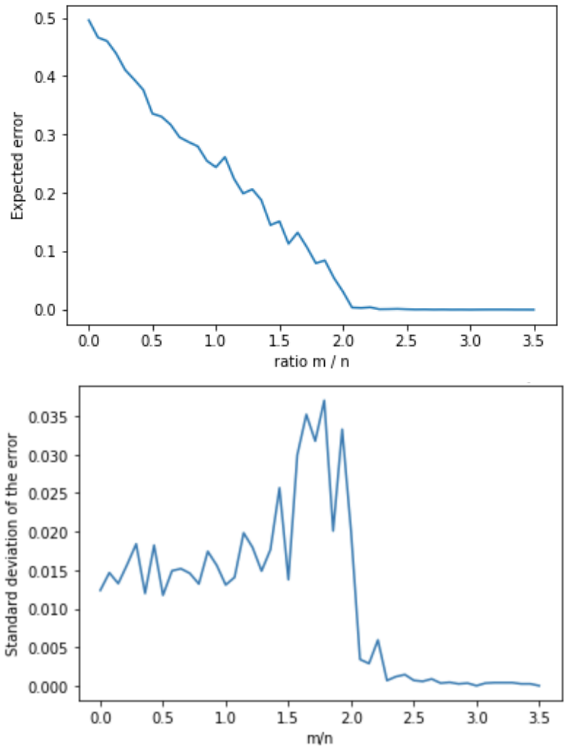

Fig. 4: Evolution of the mean and standard deviation of the error as a function of $\frac{m}{n}$, results on 50 different ratios

and have to work backwards a bit to reach a small enough error. This yields high sandard deviation. Eventually as all of the experiments reach the wanted vector X, the standard deviation will converge to 0.

V. CONCLUSIONS

This project gave us a better insight on linear estimation and especially for non-time homogeneous markov chains. We enjoyed the challenge of fast convergence as well as the interpretation and analysis of results