TP 1 Inverse Problem
Gibbs Sampler

Cabeza Gallucci, Manuel
mcabezag@fi.uba.ar

February 26, 2023

Contents

1 Introduction

The objective is to find the sparse signal $s \in \mathbb{R}^D$ from the observations $x \in \mathbb{R}^N$ with the following model of added Gaussian noise. $A \in \mathbb{R}^{NxD}$ (N«D).

$$x = As + n$$

To impose sparsity we define s as a Bernoulli-Gaussian using the binary sequence $q = [q_1 ... q_D]^T$ and L = $\sum_{d=1}^{D} q_d$. Thus, the amplitudes s_i are assumed iid zero-mean Gaussians as:

$$s|q \sim N(0, \sigma_s^2 diag(q))$$
$$P(q) = \lambda^L (1-\lambda)^{D-L}$$

We then look to estimate $\Theta = s, q, \lambda, \sigma_n^2, \sigma_s^2$ from the observations x with λ following a Beta distribution.

According to the Monte Carlo principle we can estimate Θ as seen below, so that the samples converge asymptotically to the probability given below.

$$\hat{\Theta} = \frac{1}{I-J} \sum_{k=J+1}^{I} \Theta_k$$

$$P(\Theta|x) \propto g(x - As, \sigma_n^2 I_N) P(q, \lambda) g(s, \sigma_s^2 diag(q)) P(\sigma_n^2) P(\sigma_s^2) P(\lambda)^1 \quad (1)$$

[1] Here g(μ,R) is the Gaussian distribution

2 Algorithm implementation

2.1 Conditional probability

We aim to find the conditional probabilities needed in the Gibbs sampler. Firstly, (1) is derived from the assumptions,

$$P(\Theta|x) \propto \frac{1}{(2\pi\sigma_n^2)^{N/2}} \exp[-\frac{1}{2\sigma_n^2}||x-As||^2]\lambda^{\sum q_d}(1-\lambda)^{D-\sum q_d}\frac{1}{((2\pi)^D|\sigma_s^2 diag(q)|)^{1/2}} \exp[-\frac{1}{2\sigma_s^2}s^T diag(q)s] \quad (2)$$

Then, to find the probability of q_d we consider that it can only take the values 0, 1 and we take the joint probability with s_d. Given that $s_d|_{qd=0} = 0$ we can write the distribution as a Dirac $\delta(s_d)$. That way, integrating over s_d becomes trivial. This yield the expression for q_d conditioned to $\Theta\backslash\{q_d, s_d\}$ and the observations.

$$P(q_d = 0, s_d|x, \Theta\backslash\{q_d, s_d\}) \propto (1-\lambda) \exp[\frac{-1}{2\sigma_n^2}||x - As||^2]\delta(s_d)$$

$$P(q_d = 0|x, \Theta\backslash\{q_d, s_d\}) \propto (1-\lambda) \exp[\frac{-1}{2\sigma_n^2}||x - A_{-d}s_{-d}||^2]$$

We note the matrix A_{-m} as the matrix without the m-th column. For $q_d = 1$ we get that the conditioned probability $s_d|_{qd=1}$ is equal to a centred normal with variance σ_s^2.

$$P(q_d = 1, s_d|x, \Theta\backslash\{q_d, s_d\}) \propto \lambda \exp[\frac{-1}{2\sigma_n^2}||x - As||^2]\frac{1}{(2\pi\sigma_s^2)^{1/2}} \exp[-\frac{s_d^2}{2\sigma_s^2}]$$

We can then marginalize this probability, given that $s_d \sim N(\mu_d, \sigma_d^2)$,

$$P(q_d = 1|x, \Theta\backslash\{q_d, s_d\}) \propto \lambda \exp[\frac{-1}{2\sigma_n^2}||x - A_{-d}s_{-d}||^2]\frac{\sigma_n}{\sigma_s^2} \exp[-\frac{\mu_d^2}{2\sigma_d^2}]$$

With $\mu_d = \frac{\sigma_d^2}{\sigma_n^2}A_n^T(x - A_{-d}s_{-d})$ and $\sigma_d^2 = \frac{\sigma_n^2\sigma_s^2}{\sigma_n^2+\sigma_s^2||A_n||^2}$.

Then, if we define $\gamma_d = \lambda\frac{\sigma_n}{\sigma_s} \exp[-\frac{\mu_d^2}{2\sigma_d^2}]$ we find that the conditional probability of q follows the Bernoulli distribution.

$$q_n|\Theta\backslash\{q_d, s_q\}, x \sim B(\frac{\gamma_d}{\gamma_d + 1 - \lambda})$$

As said before, s_d is equal to 0 if $q_d = 0$ or it follows a Normal distribution, thus we have the probability,

$$s_d|\Theta\backslash\{s_q\}, x : \begin{cases} 0 & \text{if } q_d = 0 \\ N(\mu_d, \sigma_d^2) & \text{if } q_d = 1 \end{cases}$$

In the case of λ, we observe from the definition that it comes a Beta distribution whose parameters depend on L and D.

$$\lambda|\Theta\backslash\{\lambda\}, x \propto \lambda^{\sum q_d}(1-\lambda)^{D-\sum q_d}$$

$$\lambda|\Theta\backslash\{\lambda\}, x \sim Beta(1 + L, 1 + D - L)$$

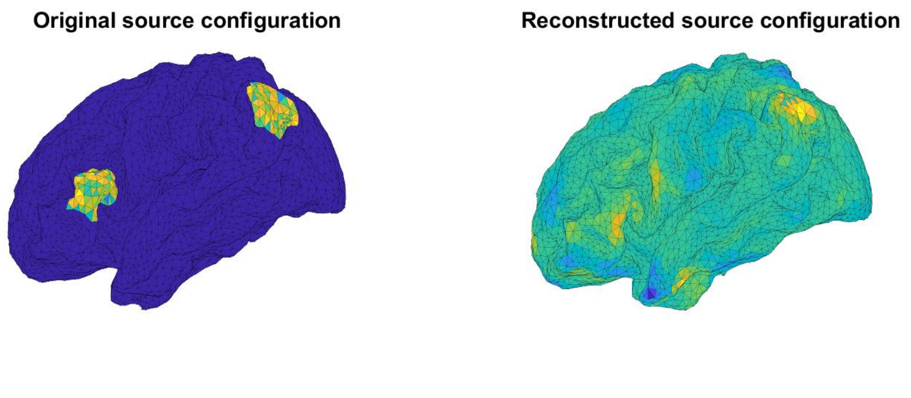
2.2 Algorithm

The algorithm was implemented in Matlab following the given pseudocode. The implementation is in Annex I.

3 Results

3.1 Data

The results are based on the active dipoles problem, where the vector s represents the sources from all points and x the measured points in the brain. The objective is to find the correct source configuration for an epileptic strike. In all cases 100 iterations of the algorithm are performed. For this problem we have $N = 91$ and $D = 19626$.


Original source configuration
Reconstructed source configuration

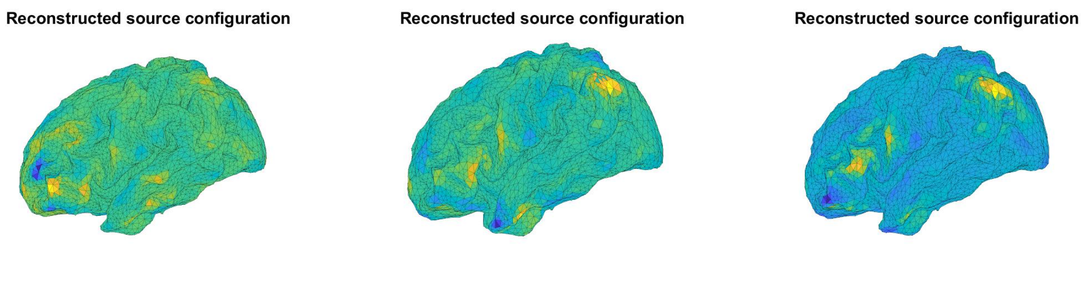
(a) Original sparse signal (two sources). (b) Results after 100 iterations.

Figure 1: Results obtained for SNR=1.

We observe that the solution is good, since the solution is sparse and concentrates most of its energy around the sources. Notably, the algorithm took a long time to execute, this is most likely due to one of the dimensions being so large.

3.2 SNR variation

Now the SNR is changed from 0.1 to 10 in order to see the effects on the results. We observe from the following figure that when the noise is high (low SNR) the result is less sparse and more noisy. As the SNR goes up we get a solution that approaches the real problem sources. As expected, we see most of the energy of the signal around the two sources.


Reconstructed source configuration
Reconstructed source configuration
Reconstructed source configuration

(a) SNR=0.1. (b) SNR=1. (c) SNR=10.

Figure 2: Results for different values of SNR.

4 Conclusion

The Gibbs sampling algorithm, which works by making hypothesis on the data to ensure an sparse solution was analyzed and implemented in Matlab. A practical exampled was shown in the case of a brain signal for an epileptic strike. We observe that when then SNR is not too small (bigger than 1), the results are quite close and both sources are found. Also, the execution times are quite long.

5 Annex I. Code

```
function [SOut,LambdaOut]=Gibbs_sampler(X,A,sigma2)

%number of iterations
Niter=100;
Nsearch = Niter/2;

%get number of sensors and number of dipoles
[N,D]=size(A);

% constants
nA = sum(A.^2,1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% implement the Gibbs sampler here according to the pseudocode
% store vectors q and s of each iteration in matrices Q and S (D x niter)
% use variables sigma_n2 and sigma_s2 for the variances of noise and s
Q = zeros(D,Niter);
S = zeros(D,Niter);
lambda = betarnd(1,1);
sigma_n2 = sigma2;
sigma_s2 = sigma2;
for k = 1:Niter
    fprintf('Iteration: %d \n',k);
    e = X - A * S(:,k);
    for i = 1:D
        e_i = e + A(:,i)*S(i,k);
        sigma_i2 = sigma_n2 * sigma_s2 / (sigma_n2 + sigma_s2 * nA(i) );
        mu_i = (sigma_i2 / sigma_n2) * A(:,i)' * e_i;
        v_i = lambda * sqrt(sigma_i2 / sigma_s2) * exp(mu_i^2 / (2*sigma_i2));
        if v_i > 1e10
            lambda_i = 1;
        else
            lambda_i = v_i / (v_i + 1 - lambda);
        end
        Q(i,k) = binornd(1,lambda_i, 1, 1);
        S(i,k) = Q(i,k) .* normrnd(mu_i, sqrt(sigma_i2)); %((randn(1,1) * sqrt(
            sigma_i2) + mu_i));
        e = e_i - A(:,i)*S(i,k);
    end
    L = sum(Q(:,k));
    lambda = betarnd(1+L,1+D-L);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Make the estimation from Q and S using the MAP criterium
%Q(:,Niter/2:Niter)
cout = zeros(Nsearch,1);
for j = 1:Nsearch
    q = Q(:,Niter-Nsearch+j);
    idx = find(q==1);
    R = (A(:,idx)'*A(:,idx))/sigma_n2 + eye(length(idx))/sigma_s2;
    S_opt = (R\(A(:,idx)'*X))/sigma_n2;
    cout(j) = - S_opt'*R*S_opt/sigma_n2;
end

[~, OptIdx] = min(cout);
q = Q(:,Niter/2+OptIdx);
idx = find(q==1);
R = (A(:,idx)'*A(:,idx))/sigma_n2 + eye(length(idx))/sigma_s2;
SOut = zeros(D,1);
SOut(idx) = (R\(A(:,idx)'*X))/sigma_n2;
LambdaOut = length(idx)/D;
```