Course mini-project

The aim of the present project is to code and experiment the Markov Chain Monte-Carlo (MCMC) method for a non-linear estimation problem. One component of the project is to get familiar with the idea of *simulated annealing*, where the temperature parameter is lowered progressively.

This is an open project were you have to experiment in order to improve the performance of the method.

Non-linear estimation problem

One would like to recover a vector $X \in S = \{-1,+1\}^N$ from the following non-linear noisy observations:

$$Y_{ij} = \sqrt{\frac{\lambda}{N}}\, X_i\, X_j + Z_{ij}, \quad 1 \le i < j \le N$$

where $\lambda > 0$ is some fixed parameter and $(Z_{ij},\, 1 \le i < j \le N)$ are i.i.d.$\sim \mathcal{N}(0,1)$ random variables. In matrix form, this reads:

$$Y = \sqrt{\frac{\lambda}{N}}\, X X^T + Z$$

As the problem is non-linear and the search space is large (of size 2^N), it is not clear how to solve this problem efficiently. We propose here to approach the problem via the MCMC method.

For a given observation Y, we define the *energy* of a vector $x \in S = \{-1,+1\}^N$ as

$$H_Y(x) = \sum_{i<j} \left| Y_{ij} - \sqrt{\frac{\lambda}{N}}\, x_i\, x_j \right|^2 \tag{1}$$

Our aim is to minimize this energy via the MCMC method. For this purpose, we introduce an auxiliary *finite temperature* version of the problem. Consider the *Gibbs-Boltzmann* probability distribution

$$p_Y(x) = \frac{e^{-\beta H_Y(x)}}{Z_\beta}, \quad x \in S$$

where $\beta = 1/T$ is the *inverse temperature* and $Z_\beta = \sum_{x \in S} e^{-\beta H_Y(x)}$ is the normalizing factor (also called the *partition function*). The idea described below is to construct a Markov chain that samples correctly from this distribution. As we take the parameter $\beta \to +\infty$ (equivalently $T \to 0$), the obtained sample should converge to a vector x that minimizes the above energy function (1).

Metropolis chain

For the moment, think of $\beta > 0$ fixed. For this, we design the following Metropolis chain on the state space S:

1. Start from a random initial vector $x^{(0)} \in S$;
2. The base chain has the following transition mechanism: at each step, choose a coordinate i of the current vector $x^{(0)} \in S$ uniformly at random and flip it, so that $x^{(1)} = (x_1, \ldots, x_{i-1}, -x_i, x_{i+1}, \ldots, x_N)$. (*NB:* Does this base chain satisfy the assumptions of the theorem seen in class?)
3. Accept then the move with probability
$$a_\beta(x^{(0)}, x^{(1)}) = \min(1, \exp(-\beta(H_Y(x^{(1)}) - H_Y(x^{(0)}))))$$
4. Iterate point 2 to obtain the Markov chain $x^{(t)}$, $t = 0, \ldots, T$ until the energy $H_Y(x^{(T)})$ is sufficiently low.

All this with the hope that a state x of low energy $H_Y(x)$ is as close as possible to the original vector X. As a justification for this, please check that the vector x minimizing the energy $H_Y(x)$ corresponds to the one maximizing the log-likelihood function: $\log(p(Y|X = x))$. [! not the same p as above !]

An alternate algorithm: Glauber or heat bath dynamics

1. Start from a random initial vector $x^{(0)}$ at time $t = 0$.
2. Select a vertex i at random from $\{1, \cdots, N\}$, erase the value of x_i, and reset this value to ± 1 with probabilities
$$p_\pm = \frac{1}{2}\left(1 \pm \tanh\left(2\beta\sqrt{\frac{\lambda}{N}}\sum_{\substack{j=1\\ j\neq i}}^{N} Y_{ij}x_j\right)\right)$$
This yields a new vector $x^{(1)}$ (the derivation of this formula is exactly the same as for the Ising model seen in class; try it explicitly!).
3. Iterate point 2 to obtain the Markov chain $x^{(t)}$, $t = 0, \ldots, T$ until the energy $H_Y(x^{(T)})$ is sufficiently low.

You are free to choose the algorithm (Metropolis or Glauber) for your implementation. An even better option is to try both and opt for the best!

Simulated annealing

The main idea in simulated annealing is to lower the temperature $T = \frac{1}{\beta}$ as time goes by. Note that the transition probabilities change with time now, so the Metropolis chain is not anymore time-homogeneous.

You will have to experiment to find a suitable schedule for lowering T (resp., increasing β). If you lower T too slowly, you might not reach a minimizer of $H_Y(x)$ in a decent amount of time. If you lower T too quickly, you lose the advantage of the Metropolis step and might end up in a local minimum with a high energy.

One (basic) advice is not to change the temperature at every iteration step, but rather to keep it constant for some steps before lowering the temperature.

Some important questions and remarks

- Do you need to know the value of the parameter λ in order to implement the algorithm?
- What is the minimal value of the energy you can hope for?
- Compared to the graph coloring problem seen in class, you should observe here that the proposed moves in the chain have quite a large impact on the total energy of the system (as the graph is *complete* here). This makes the algorithm slightly less stable than in the case of a classical sparse graph (where each vertex is connected, say, to a constant number of vertices). As a consequence, multiple runs of the algorithm starting from various initial conditions are beneficial here.
- A perfectly valid question, now: what are the applications of this problem? Two main applications are:
  - the above estimation problem can be mapped into a clustering problem;
  - it falls into the larger category of low-rank matrix estimation problems (here, the matrix XX^T is rank one).

What we expect from you

Your tasks in this project are the following:

1. Form teams of 3 to complete this project, choose a team name and send it to us by email by **Wednesday, November 23.**

2. Implement a version of the above MCMC method, along with simulated annealing, which finds, for a given observation Y, a vector $\hat{x}$ with energy $H_Y(\hat{x})$ as low as possible. Your code should be preferably in Matlab. (*NB:* The larger the value of N, the better!)

3. Describe and optimize your “cooling strategy” for simulated annealing, and plot the energy obtained by your algorithm as a function of time.

4. Estimate the (normalized) *mean energy* $\mathbb{E}(\frac{1}{N}H_Y(\hat{x}))$ reached by your algorithm, by running the whole thing multiple times for various realizations of Y (corresp., Z). Draw the result as a function of the parameter $\lambda > 0$.

5. Estimate the (normalized) *mean squared error* $\mathbb{E}(\frac{1}{N}\sum_{i=1}^{N}(\hat{x}_i - X_i)^2)$ reached by your algorithm, by running the whole thing multiple times for various realizations of Y. Draw the result as a function of the parameter $\lambda > 0$.

6. Is there a limiting value $\lambda_c > 0$ below which your algorithm does not perform better than a purely random guess?

You should then

- produce a small report (2-4 pages) answering the above questions, as well as a short description of your code **(due Wednesday, December 14)**;

- be prepared to participate to the final competition on **Wednesday, December 21, at 2:15 PM!**