Simulation and Monte Carlo

2022-2023 projects

Nicolas Chopin

You must form a group of three students among your TD class, select one of the following projects, and do it by the last TD class, where you will do an oral presentation of your results in front of the other students. You may return a document with your plots and results to your "chargé(e) de TD" (no need to return a formal report). Please make sure you also send your code.

Important: Always make sure you have assessed in some way the numerical error of your estimates (e.g. through box-plots over repeated runs for instance).

Bonus questions are **optional**: do them when are confident you treated properly the previous points.

It's ok to look up online for solutions (and adapt code you found on the internet), but cite your sources accordingly.

If you are stuck, feel free to get in touch with your "chargé(e) de TD", who will let me know.

ABC for Lotka-Volterra models

The following paper by Lee and Łatuszyński https://doi.org/10.1093/biomet/asu027 proposes an MCMC algorithm for likelihood-free inference (i.e. ABC, see final chapter).

Try to reproduce the numerical experiment of Section 4.3 in this paper. In order to so,

1. Simulate data from the model (using the same parameter values as in the paper), using Gillespie's algorithm. (This algorithm is explained on Wikipedia for instance, or in various scientific papers; it is also strongly related to the point processes presented in the last chapter of the poly of "Introduction aux processus").

2. Implement and compare Algorithms 1 and 3 in the paper, for various values of N (you may start with $N = 1$, consider the first prior discussed in the paper, and the same pseudo-distance). Try to think about (a) ways to represent graphically the relative performance of the algorithms; and (b) why Algorithm 3 may perform better than Algorithm 1.

3. If you have time, you can Algorithm 2 and/or Algorithm 4 to the comparison, and try to think on how to choose N. You can consider the two prior distributions discussed in the paper.

Russian roulette

This project is based on the first numerical experiment of this paper: https://doi.org/10.1214/15-STS523, which recovers the posterior distribution of parameter β, for an Ising model.

1. Implement a Gibbs sampler to sample from the model (6.1), using the same settings as in the paper (same values for α, β, same 10×10 grid, etc.).

2. Implement an exchange algorithm to sample from the same posterior distribution as in the paper; for a reference on the exchange algorithm, see for instance the paper by Murray et al here: https://arxiv.org/abs/1206.6848. Explain why you need this type of algorithm for this model, how the exchange algorithm works, and how you may use it in your case (using Step 1).

   Note: you cannot sample *exactly* from the model using the Gibbs sampler of Step 1; however, you can run the Gibbs sampler for a large number of iterations to sample *approximately* from the model. Try to assess numerically the impact on the number of Gibbs steps you use to sample from the model (at each step of the exchange algorithm) on the performance of the algorithm.

3. Bonus: compare with the Russian roulette approach proposed in the paper, and discuss.

Control variates for a GARCH model

This project is based on the numerical example on GARCH modelling from this paper: https://doi.org/10.1007/s11222-012-9344-6.

1. Follow the guidelines given in that experiment to implement a random walk Metropolis sampler that targets the posterior distribution of a GARCH model. For the data, you can use simulated data at first, and then look at the same type of real data (log-returns computed from exchange rates) in a second time.

2. Consider the control variates proposed in the paper (Equation (8), and in particular the versions based on a first-order polynomial, i.e. $f(x) + a^T z$ on the same page as (8)). Explain how you can you use these control variates in this settings by doing a linear regression.

3. Consider a larger set of control variates (by using a polynomial of order 2 for instance). When the number of control variates become too large, the linear regression approach of Step 2 may become too expensive (explain why). Adapt the method proposed in this paper: https://doi.org/10.1007/s11222-021-10011-z, which relies on the Lasso, to your problem, and compare with the naive approach (where you use only a single step based on a linear regression).

4. Bonus: is the fact that we do linear regression completely valid when applied to a MCMC sample? Try to think of ways to address this issue.

   Hint: you could think of ways to make MCMC simulations less "dependent", by sub-sampling, or averaging over blocks of a given size, or another approach.

Stratification

Consider the following function of $u \in [0,1]^d$, for $d \geq 1$:

$$f(u) = 1 + \sin\left(2\pi(\frac{1}{d}\sum_{i=1}^{d} u_i - \frac{1}{2})\right)$$

1. Approximate the integral of f using standard Monte Carlo and quasi-Monte Carlo, and compare the results, for different values of d (and different Monte Carlo sample sizes!).

2. Read the introduction of the following paper https://arxiv.org/abs/2210.01554 up to Equations (5) and (6), which introduce Haber's estimators of order 1 and 2. Implement these two estimators, and compare to the results of Step 1. (Again consider different values of d and N, where N is the number of evalations of f).

3. Bonus: try to think of an importance sampling approach to obtain a better estimate of this integral.

Option pricing

We wish to evaluate the price of an Asian option

$$C = \mathbb{E}\left[e^{-rT}\left(\frac{1}{k}\sum_{i=1}^{k} S(t_i) - K\right)^+\right]$$

where $t_0 = 0 < t_1 < \ldots < t_k = T$, $r > 0$, K fixed, and S a process defined over $[0, T]$; note that: $x^+ = \max(x, 0)$.

We consider the CIR (Cox, Ingersoll, Ross) model:

$$dS_t = \alpha(b - S_t)dt + \sigma\sqrt{S_t}dW_t$$

where W_t is a Brownian motion. You may take $\alpha = 0.15$, $b = 0$, $\sigma = 0.2$, $T = 1$, $r = 0.05$, $K = 4$, $k = 20$, $t_i = i/20$.

1. Explain and implement the MLMC (multi-level Monte Carlo) method presented in the final part of the course (see also http://statweb.stanford.edu/~owen/courses/362/readings/GilesMultilevel.pdf) to approximate C. Compare with standard Monte Carlo (one level only), both in terms of CPU time and in terms of MSE (mean square error). Explain carefully how you chose the Monte Carlo sample size for each level.
2. Repeat the comparison using (randomised) quasi-Monte Carlo.
3. Bonus: try to see how much the result differ if you use the Brownian bridge construction in Step 2.

   Recall that to generate the trajectory of a Brownian motion at times $t_0 \leq t_1 \leq ... \leq t_K$, you may generate sequentially (and independently) the increments $B_{t_1} - B_{t_0}$, $B_{t_2} - B_{t_1}$, etc. but can also generate the different B_{t_k} in a given order, as explained in the course, for instance based on a van der Corput sequence in base 2: assuming $K = 2^i$, sample B_K first, then $B_{K/2}$, then $B_{K/4}$ and $B_{3K/4}$ and so on. Explain how you may derive the distribution of each B_{t_k} given the ones that have already been generated, and how you may use this construction within RqMC.

Log-linear models

Log-linear models apply to a vector X taking values in $\{0, 1\}^d$. The probability of observing a certain x is such that (for a model restricted to two-order interactions):

$$\log P(X = x) = \alpha + \sum_{i=1}^{d} \beta_i x_i + \sum_{i<j} \gamma_{ij} x_i x_j$$

1. Propose and implement an algorithm to sample data from this model (for fixed values of β_i). How can you check that the corresponding algorithm mixes well in this case?
2. How is α related to the other parameters? What particular problem does this quantity pose if one wishes to estimate the parameters of this model (based on data).

3. To address the issue raised in point 2, implement the exchange algorithm presented in https://arxiv.org/pdf/1206.6848.pdf.
4. Bonus: apply what have done so far to estimate this kind of model on real datasets such as UCBAdmissions or Titanic.

Unbiased gradient

A very popular technique in Machine Learning is to maximise the likelihood of the considered model through SGD (stochastic gradient descent), which is gradient descent (applied to minus the likelihood) with the true gradient replaced by an unbiased estimate.

The following paper presents a novel method to obtain unbiased estimates of gradient for latent variable models: http://proceedings.mlr.press/v130/shi21d.html.

The objective of this project is to reproduce their first numerical experiment, and compare their two estimators (based on either the simple term estimate, or the Russian roulette one) with IWAE (which is biased). You can do the same comparison, and you can also compare the performance of SGD when using the different estimates of the gradient.

Take the time to explain what is the idea behind (7), and in particular why this may work better than just subtracting an estimate based on 2^k samples.

Bonus: if you have time, you can also SUMO (mentioned in he paper) to the comparison. You can also start to have a look on how these ideas may be used within a variational auto-encoder, and give a high-level overview of this kind of method to the rest of the class.

Tetris

Warning: this project may be more challenging and time-consuming than the others.

Following https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=b19922afc8678a228c780715d50f5a427dc51680, the objective of this project is to implement various optimisation schemes to learn how to play Tetris:

- The standard CE (cross-entropy) method (based on the family of Gaussian distributions);
- the variant proposed in the above paper, where noise is introduced at each iteration.
- Bonus: simulated annealing.

The function to optimise takes as arguments the 22 coefficients of the value function defined in (1) in the paper, and returns the number of steps the Tetris game stops (game over) when it follows the corresponding policy. (This "function" is actually random, since the order of the pieces is random.) To implement the Tetris game, feel free to use a library like this one: https://github.com/dzshn/python-tetris.