TP 4: Improve the Metropolis-Hastings algorithm

In this lab session, there are two mandatory exercises (Ex1 and Ex2) for a total of 17 points. The overall code quality will receive a grade from 0 to 3pts based off the following criterias:

- Readability: the names of the variables are easy to understand. When necessary, comments help the reader understand the code. The code is not unnecessarily long or complex.
- Efficiency: functions are defined when the same code is used several times (no copy-paste). The code is flexible; it is easy to change a parameter.

Exercise 1: Adaptive Metropolis-Hastings within Gibbs sampler (9pt)

MCMC samplers, such as Metropolis-Hastings or Gibbs samplers, require that the user specify a transition kernel given a target distribution. These transition kernels usually depend on parameters which are to be given and tuned by the user. In practice, it is often difficult (if not impossible) to find the best parameters for such algorithms given a target distribution. If the parameters are not carefully chosen, it may result in a MCMC algorithm performing poorly. In this exercise, we explore *Adaptive MCMC algorithms* which address the problem of parameters tuning by updating automatically some of the parameters.

1.A – Metropolis-Hastings within a Random Scan Gibbs sampler

We aim to sample the target distribution π, on $\mathbb{R}^2$, given by

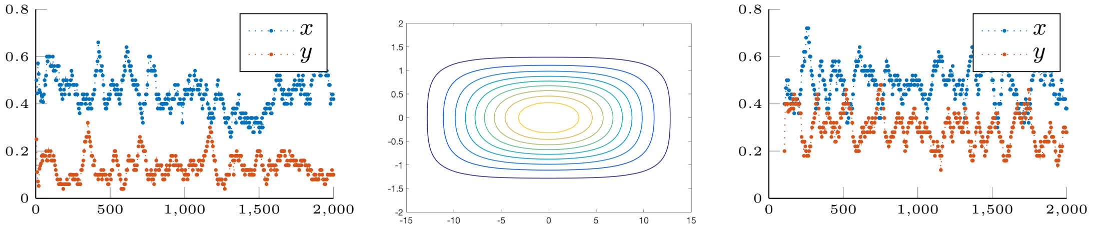
$$(x, y) \mapsto \pi(x, y) \propto \exp\left(-\frac{x^2}{a^2} - y^2 - \frac{1}{4}\left(\frac{x^2}{a^2} - y^2\right)^2\right)$$

where $a > 0$. We consider a Markov transition kernel P defined by

$$P = \frac{1}{2}(P_1 + P_2)$$

where $P_i(\,(x, y)\,;\, \mathrm{d}x' \times \mathrm{d}y'\,)$ for $i = 1, 2$ is the Markov transition kernel which only updates the i-th component: this update follows a symmetric random walk proposal mechanism and uses a Gaussian distribution with variance σ_i^2.

1. Explain why this MCMC algorithm is called a Random Scan Gibbs sampler. When is Metropolis-Hasting used in this sampler ? (1pt)
2. (Python) Implement an algorithm which samples the distribution $P_1(z; \cdot)$ where $z \in \mathbb{R}^2$; likewise for the distribution $P_2(z; \cdot)$. Then, implement an algorithm which samples a markov chain with kernel P. (2pt)
3. (Python) Run the algorithm with $a = 10$ and standard deviations of the proposal distributions chosen as follows: $(\sigma_1, \sigma_2) = (3, 3)$. Discuss the performance of the algorithm in this situation. (1pt)
4. How could the performance of the above algorithm be improved ? (1pt)



Acceptance rate, HM sampler

Acceptance rate, adaptative-HM sampler

Figure 1: Mean acceptance rate and contour plot of the density $-\ a = 10$

1.B – Adaptive Metropolis-Hastings within Gibbs sampler

(9pt)

We now consider a Gibbs sampler with a fixed-order scan of the coordinates (as seen in the lecture) to sample from any distribution π on $\mathbb{R}^d$. We still consider Metropolis-Hastings steps to sample from the conditional distributions. As usual, for $i \in [\![1, d]\!]$, let π_i denote the i-th full conditional of π, which is given by:

$$x_{-i} = \{x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_d\} \qquad ; \qquad \pi_i(x_i \,|\, x_{-i}) \propto \pi(x)$$

The MH-steps use a symetric random walk with a gaussian proposal distribution with variance σ_i^2 for the i-th coordinate update. An entire Gibbs sampler update is recalled in Algorithm 1.

Algorithm 1: Metropolis-Hastings (symmetric random walk) within Gibbs Sampler

```
Given $x^{(k)} = (x_1^{(k)}, \dots, x_d^{(k)})$
**for** $i = 1$ *to* $d$ **do**
    HM to sample from the target $x_i^{(k+1)} \sim \pi_i(x_i \mid x_1^{(k+1)}, \dots, x_{i-1}^{(k+1)}, x_{i+1}^{(k)}, \dots, x_d^{(k)})$:
        Proposal: $x_i^* \sim \mathcal{N}(x_i^{(k)}, \sigma_i^2)$
        Acceptance ratio $\alpha(x_i^*, x_i^{(k)}) = \frac{\pi_i(x_i^* \mid x_1^{(k+1)}, \dots, x_{i-1}^{(k+1)}, x_{i+1}^{(k)}, \dots, x_d^{(k)})}{\pi_i(x_i^{(k)} \mid x_1^{(k+1)}, \dots, x_{i-1}^{(k+1)}, x_{i+1}^{(k)}, \dots, x_d^{(k)})} \wedge 1$
**end**
```

In [RR09], the authors propose an adaptive version of the above sampler which automatically adjusts the variances $\sigma_1^2, \dots, \sigma_d^2$ of the proposal distributions. We proceed as follows:

- For each of the variables x_i, we create an associate variable ℓ_i giving for the logarithm of the standard deviation σ_i to be used when proposing a normal increment to variable: $\ell_i := \log(\sigma_i)$;
- We initialize all ℓ_i to zero, which correspond to the unit proposal variance ;
- After the j-th ($j \in \mathbb{N}^*$) batch of 50 iterations, each variable ℓ_i is updated by adding or subtracting an amount $\delta(j)$. The adapting attempts to make the acceptance rate of proposals for variable x_i as close as possible to 0.234, which is optimal for one-dimensional proposals in certain settings. Specifically, if the acceptance rate for the i-th variable is greater than 0.234 [GGR97],

(see `http://www.stat.columbia.edu/~gelman/research/published/A06-109-new_version.pdf` for more details) ℓ_i is increased with $\delta(j)$. Otherwise, if the rate is lower than 0.234, ℓ_i is decreased by $\delta(j)$.

In practice, we (can) take $\delta(j) := \min(0.05, j^{-1/2})$.

1. (Python) Implement the adaptative Metropolis-Hastings within Gibbs sampler and test the algorithm on the density π defined in the part A. Using auto-correlation plots (use a built-in function), compare the performance of the algorithm with or without adaptation. (3pt)

2. (Python) We can also compare the performance of our algorithm on more complicated target densities. For example centered d-dimensional Gaussian $\mathcal{N}(0, \Sigma)$ or "banana"-shaped density as in TP 2:

$$\forall x = (x_1, \ldots, x_d) \in \mathbb{R}^d, \quad f_B(x) \propto \exp\left(-\frac{x_1^2}{2} - \frac{1}{2}\big(x_2 + Bx_1^2 - B\big)^2 - \frac{1}{2}\big(x_3^2 + \ldots + x_d^2\big)\right).$$

In practice, you can choose $d = 5$ and $B = 0.4$ for the density f_B. (1pt)

To go further...

The next improvement of the Metropolis-Hastings algorithm we can make is to consider a *drift* function in the proposal distribution. Given a positive definite matrix Λ and a scale parameter $\sigma > 0$, we consider a proposal distribution of the form:

$$q_{\sigma,\Lambda}(y \mid x) = \frac{1}{(\sigma\sqrt{2\pi})^d} \frac{1}{\sqrt{\det(\Lambda)}} \exp\Big(- \frac{1}{2\sigma^2}\big[y - x - \frac{\sigma^2}{2}\Lambda D(x)\big]^\top \Lambda^{-1} \big[y - x - \frac{\sigma^2}{2}\Lambda D(x)\big]\Big).$$

$q_{\sigma,\Lambda}$ is the density (with respect to the Lebesgue measure on $\mathbb{R}^d$) of the d-dimensional Gaussian distribution with mean $x + \frac{\sigma^2}{2}\Lambda D(x)$ and variance-covariance matrix $\sigma^2\Lambda$. If D vanishes everywhere ($D \equiv 0$), the corresponding algorithm is a *Metropolis-Hastings Symmetric Random Walk*. If the drift D is chosen such that:

$$\forall x \in \mathbb{R}^d, \quad D(x) = \frac{\delta}{\max(\delta, \|\nabla \log \pi(x)\|)} \nabla \log \pi(x)$$

for a constant $\delta > 0$, the corresponding algorithm is a *Metropolis Adjusted Langevin Algorithm* (MALA). In that case, the proposal distribution includes information on the gradient $\nabla \log \pi$ of the target distribution π. In [Atc06], the author proposes an adaptive version of the MALA algorithm in which the parameters σ and Λ are adjusted automatically.

Exercise 2: Sampling from multimodal distributions (8pt)

We consider a target distribution π with support $\mathcal{U} \subset \mathbb{R}^d$ ($d \in \mathbb{N}^*$). When the target distribution is multimodal, especially with well-separated modes, classical MCMC algorithms can perform very poorly and exhibit poor mixing. Indeed, a Metropolis-Hastings algorithm with local proposal can get stuck for a long time in a local mode of the target distribution.

2.A – A toy example

In the following, we consider a target distribution π – taken from [LW01] and plotted at figure 2 – defined on $\mathbb{R}^2$ as a mixture of 20 Gaussian distributions. The target distribution writes:

$$\pi(\mathbf{x}) = \sum_{i=1}^{20} \frac{w_i}{2\pi\sigma_i^2} \exp\Big(-\frac{1}{2\sigma_i^2}\,{}^t(\mathbf{x}-\boldsymbol{\mu}_i)(\mathbf{x}-\boldsymbol{\mu}_i)\Big)$$

where, $\forall i \in \{1, \dots, 20\}$, $w_i = 0.05$ and $\sigma_i = 0.1$. The 20 means $\boldsymbol{\mu}_i$ are defined as follows:

$$\begin{aligned}(\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_{20}) \quad = \quad \Bigg(& \begin{pmatrix}2.18\\5.76\end{pmatrix}, \begin{pmatrix}8.67\\9.59\end{pmatrix}, \begin{pmatrix}4.24\\8.48\end{pmatrix}, \begin{pmatrix}8.41\\1.68\end{pmatrix}, \begin{pmatrix}3.93\\8.82\end{pmatrix}, \begin{pmatrix}3.25\\3.47\end{pmatrix}, \begin{pmatrix}1.70\\0.50\end{pmatrix}, \\ & \begin{pmatrix}4.59\\5.60\end{pmatrix}, \begin{pmatrix}6.91\\5.81\end{pmatrix}, \begin{pmatrix}6.87\\5.40\end{pmatrix}, \begin{pmatrix}5.41\\2.65\end{pmatrix}, \begin{pmatrix}2.70\\7.88\end{pmatrix}, \begin{pmatrix}4.98\\3.70\end{pmatrix}, \begin{pmatrix}1.14\\2.39\end{pmatrix}, \\ & \begin{pmatrix}8.33\\9.50\end{pmatrix}, \begin{pmatrix}4.93\\1.50\end{pmatrix}, \begin{pmatrix}1.83\\0.09\end{pmatrix}, \begin{pmatrix}2.26\\0.31\end{pmatrix}, \begin{pmatrix}5.54\\6.86\end{pmatrix}, \begin{pmatrix}1.69\\8.11\end{pmatrix} \Bigg).\end{aligned}$$

You can download these means as a numpy array 'mu.npy' on the Google Classroom.

1. (Python) Implement a function to compute $\pi(x)$ and $\log(\pi(x))$. (0.5pt)
2. (Python) Write a Metropolis-Hastings Symmetric Random Walk algorithm (you may use your code from previous tutorial classes) to sample from π. (1.5pt)
3. (Python) Show that the Metropolis-Hastings algorithm (even the adaptive Metropolis-Hastings algorithm) fails to sample from π. (1 pt)

2.B – Parallel Tempering

The general idea of the Parallel Tempering (PT) [Gey91, ED05] algorithm is to use *tempered* versions of the distribution π and run parallel Metropolis-Hastings algorithm to sample from these tempered distributions. The tempered distributions are obtained by "warming up" the target distribution π at

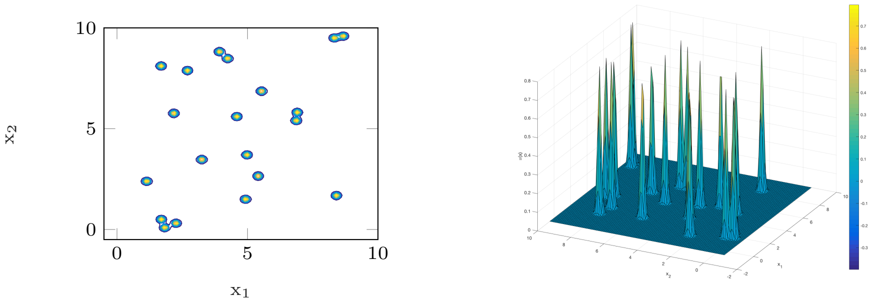


Figure 2: Mixture of 20 Gaussian distributions

different *temperatures*. At each iteration of the algorithm, a *swap* between two chains (chains running at different temperature levels) is proposed. The Parallel Tempering uses the fast mixing of the chains at high temperature to improve the mixing of the chains at low temperatures.

Let K denote a positive integer. We consider a sequence of temperatures $(T_i)_{1\leq i\leq K}$ such that:

$$T_1 > T_2 > \ldots > T_K = 1\,.$$

In the Parallel Tempering algorithm, K chains run in parallel: for $i \in [\![1, K]\!]$, the i-th chain targets the tempered distribution $\pi_i := \pi^{1/T_i}$; the distribution of interest corresponds to the lowest temperature, $T_K = 1$. Let $(X_n^{(i)})_{n\in\mathbb{N}}$ denote the i-th chain, sampling from the tempered distribution π_i.
At the n-th iteration of the Parallel Tempering algorithm, a candidate $Y_{n+1}^{(i)}$ for the i-th chain is proposed using the transition kernel $P^{(i)}(X_n^{(i)}, \cdot)$ of a Metropolis-Hastings algorithm. The next step consists in proposing a swap between two different chains (running at different temperatures): given $(i, j) \in [\![1, K]\!]^2$, with $i \neq j$, a swap is proposed with probability $\alpha(i, j)$.

Algorithm 2: Parallel Tempering

```
For all $i \in [\![1, K]\!]$, initialize $X_0^{(i)}$ ;
for $n = 1$ to $\mathrm{N}_{\mathrm{iter}}$ do
    For all $i \in [\![1, K]\!]$, draw $Y_{n+1}^{(i)}$ using the transition kernel $P^{(i)}(X_n^{(i)}, \cdot)$ ;
    Choose uniformly $(i, j) \in [\![1, K]\!]^2$, with $i \neq j$ ;
    Compute the swap acceptance probability $\alpha(i, j) = \min\left(1\,,\, \frac{\pi_i(Y_{n+1}^{(j)})\,\pi_j(Y_{n+1}^{(i)})}{\pi_i(Y_{n+1}^{(i)})\,\pi_j(Y_{n+1}^{(j)})}\right)$ ;
    Draw $U \sim \mathcal{U}([0, 1])$ ;
    if $U \leqslant \alpha(i, j)$ then
        $X_{n+1}^{(i)} = Y_{n+1}^{(j)} \quad and \quad X_{n+1}^{(j)} = Y_{n+1}^{(i)}$ ;
    else
        $X_{n+1}^{(i)} = Y_{n+1}^{(i)} \quad and \quad X_{n+1}^{(j)} = Y_{n+1}^{(j)}$ ;
    end
    For all $k \in [\![1, K]\!]$, $k \neq i, j$, set $X_{n+1}^{(k)} = Y_{n+1}^{(k)}$ .
end
```

1. Why is it easier to sample from π^{1/T_1} ? (1pt)

2. (Python) Implement the Parallel Tempering algorithm.(3pt)

3. (Python) In order to illustrate the performance of the algorithm, use your code to sample from the distribution π of Part A. Use the algorithm with $K = 5$ and with the following temperatures ladder:

$$(T_1, \ldots, T_5) = (\,60,\ 21.6,\ 7.7,\ 2.8,\ 1\,)\,.$$

For the Metropolis-Hastings step (line 3), take as proposal distribution the bivariate Gaussian distribution centered at $X_n^{(i)}$, with variance-covariance matrix $\tau_i^2 \mathrm{I}_2$:

$$\forall i \in [\![1, K]\!], \qquad Y_{n+1}^{(i)} \sim \mathcal{N}_{\mathbb{R}^2}(X_n^{(i)}, \tau_i^2 \mathrm{I}_2)\,.$$

For the scale parameters τ_i, you can try either $\tau_i = 1$ for all i or $\tau_i = 0.25\sqrt{(}T_i)$. (1pt)

In practice, the performance of the Parallel Tempering algorithm strongly depends on the choice of the temperatures ladder, the number of chains and the choice of proposal kernels. For most distributions, tuning these parameters may be infeasible. In [MMV13], the authors have proposed an adaptive Parallel Tempering algorithm to address these difficulties.

Exercise 3: Bayesian analysis of a one-way random effects model (Bonus - 3pt)

We recall that the density of an *Inverse Gamma* distribution with positive parameters (a, b) is proportional to

$$x \mapsto \frac{1}{x^{a+1}} \exp\left(-\frac{b}{x}\right) \mathbb{1}_{\mathbb{R}^+}(x)$$

and especially that we can sample y from the inverse gamma distribution of parameters (a, b) by generating x from a gamma distribution of parameters $(a, \frac{1}{b})$ and then taking $y = \frac{1}{x}$.

We can use directly `scipy.stats.invgamma` function

Let suppose we collect the observations $Y = \{\, y_{i,j},\ i \in [\![1, N]\!],\ j \in [\![1, k_i]\!] \,\}$ and set $k := \sum_{i=1}^N k_i$ the total number of observations. Let the following random effects model:

(i) $y_{i,j}$ is a realization of the variable $Y_{i,j}$ where $Y_{i,j} = X_i + \varepsilon_{i,j}$;

(ii) The random effects $X = \{\, X_i,\ i \in [\![1, N]\!] \,\}$ are i.i.d from a Gaussian $\mathcal{N}(\mu, \sigma^2)$ and independent of the errors $\varepsilon = \{\, \varepsilon_{i,j},\ i \in [\![1, N]\!],\ j \in [\![1, k_i]\!] \,\}$;

(iii) The errors ε are i.i.d from the centred Gaussian $\mathcal{N}(0, \tau^2)$;

where (μ, σ, τ) are the unknown parameters. Bayesian analysis using this model requires specifying a prior distribution, for which we consider:

$$\pi_{prior}(\mu, \sigma^2, \tau^2) \propto \frac{1}{\sigma^{2(1+\alpha)}} \exp\left(-\frac{\beta}{\sigma^2}\right) \frac{1}{\tau^{2(1+\gamma)}} \exp\left(-\frac{\beta}{\tau^2}\right)$$

where α, β and γ are known hyper-parameters.

1. Write the density of the *a posteriori* distribution $(X, \mu, \sigma^2, \tau^2 | Y)$ — it can be given up to a normalizing constant — *i.e* the density of the distribution $(Y, X, \mu, \sigma^2, \tau^2)$. (1pt)
2. (Python) Implement a Gibbs sampler which updates in turn $(\sigma^2, \tau^2, \mu, X)$ one at a time. (1pt)
3. (Python) Generate a synthetic dataset following the previous model and test your Gibbs sampler on this dataset. (1pt)
4. (Going further) Implement a Block-Gibbs sampler which updates σ^2, then τ^2 and then the block (X, μ).

References

[Atc06] Yves F. Atchadé. An adaptive version for the metropolis adjusted langevin algorithm with a truncated drift. *Methodology and Computing in Applied Probability*, 8(2):235–254, 2006.

[ED05] David J Earl and Michael W Deem. Parallel tempering: Theory, applications, and new perspectives. *Physical Chemistry Chemical Physics*, 7(23):3910–3916, 2005.

[Gey91] Charles J Geyer. Markov chain monte carlo maximum likelihood. 1991.

[GGR97] Andrew Gelman, Walter R Gilks, and Gareth O Roberts. Weak convergence and optimal scaling of random walk metropolis algorithms. *The annals of applied probability*, 7(1):110–120, 1997.

[KTB13] D.P. Kroese, T. Taimre, and Z.I. Botev. *Handbook of Monte Carlo Methods*. Wiley Series in Probability and Statistics. Wiley, 2013.

[LW01] Faming Liang and Wing Hung Wong. Real-parameter evolutionary monte carlo with applications to bayesian mixture models. *Journal of the American Statistical Association*, 96(454):653–666, 2001.

[MMV13] Błażej Miasojedow, Eric Moulines, and Matti Vihola. An adaptive parallel tempering algorithm. *Journal of Computational and Graphical Statistics*, 22(3):649–664, 2013.

[RR09] Gareth O. Roberts and Jeffrey S. Rosenthal. Examples of adaptive mcmc. *Journal of Computational and Graphical Statistics*, 18(2):349–367, 2009.