COMPUTO

ISSN 2824-7795

Variational inference for approximate reference priors using neural networks

Nils Baillie[1] Université Paris-Saclay, CEA, Service d'Études Mécaniques et Thermiques, 91191 Gif-sur-Yvette, France

Antoine Van Biesbroeck CMAP, CNRS, École polytechnique, Institut Polytechnique de Paris, 91120 Palaiseau, France

Clément Gauchy Université Paris-Saclay, CEA, Service de Génie Logiciel pour la Simulation, 91191 Gif-sur-Yvette, France

Date published: 2025-02-07 Last modified: 2025-02-07

Abstract

In Bayesian statistics, the choice of the prior can have an important influence on the posterior and the parameter estimation, especially when few data samples are available. To limit the added subjectivity from a priori information, one can use the framework of reference priors. However, computing such priors is a difficult task in general. We develop in this paper a flexible algorithm based on variational inference which computes approximations of reference priors from a set of parametric distributions using neural networks. We also show that our algorithm can retrieve reference priors when constraints are specified in the optimization problem to ensure the solution is proper. We propose a simple method to recover a relevant approximation of the parametric posterior distribution using Markov Chain Monte Carlo (MCMC) methods even if the density function of the parametric prior is not known in general. Numerical experiments on several statistical models of increasing complexity are presented. We show the usefulness of this approach by recovering the target distribution. The performance of the algorithm is evaluated on the prior distributions as well as the posterior distributions, jointly using variational inference and MCMC sampling.

Keywords: Reference priors, Variational inference, Neural networks

Contents

1 Introduction **3**

2 Reference priors theory **4**

3 Variational approximation of the reference prior (VA-RP) **6**

3.1 Implicitly defined parametric probability distributions using neural networks 6

3.2 Learning the VA-RP using stochastic gradient algorithm 7

3.3 Adaptation for the constrained VA-RP 9

3.4 Posterior sampling using implicitly defined prior distributions 11

4 Numerical experiments **13**

4.1 Multinomial model 13

4.2 Probit model 22

[1]Corresponding author: nils.baillie@cea.fr

5 Conclusion **35**

Acknowledgement **35**

6 Appendix **35**
6.1 Gradient computation of the generalized mutual information 35
6.2 Gaussian distribution with variance parameter 36
6.3 Probit model and robustness 44

References **51**

```
import os
import sys
base_dir = os.getcwd()  # the path must be set on VA-RP
py_dir = os.path.join(base_dir, "python_files")
if py_dir not in sys.path:
    sys.path.append(py_dir)
#print(sys.path)
from aux_optimizers import *
from stat_models_torch import *
from neural_nets import *
from variational_approx import *
from div_metrics_torch import *
from constraints import *

# Packages used
from scipy.stats import gaussian_kde
import matplotlib.pyplot as plt
from matplotlib import rc
from matplotlib import gridspec
import pickle

rc('text', usetex=False)
rc('lines', linewidth=2)
rougeCEA = "#b81420"
vertCEA = "#73be4b"

path_plot = os.path.join(base_dir, "plots_data")
probit_path = os.path.join(base_dir, "data_probit")
tirages_path = os.path.join(probit_path, "tirages_probit")
int_jeffreys_path = os.path.join(probit_path, "Multiseed_AJ")
varp_probit_path = os.path.join(probit_path, "Multiseed_VARP")

### Load parameter for each main computation
# If True, loads the different results without re-doing every computation
# If False, all computations will be done and saved (and possibly overwrite the data files !)

load_results_Multinom = True
load_results_MultiMMD = True
load_results_Probit_nocstr = True
load_results_Probit_cstr = True
```

```
load_results_Normal_nocstr = True
load_results_Normal_cstr = True
```

1 Introduction

The Bayesian approach to statistical inference aims to produce estimations using the posterior distribution. The latter is derived by updating the prior distribution with the observed statistics thanks to Bayes' theorem. However, the shape of the posterior can be heavily influenced by the prior choice when the amount of available data is limited or the prior distribution is highly informative. For this reason, selecting a prior remains a daunting task that must be handled carefully. Hence, systematic methods has been introduced by statisticians to help the choice of the prior distribution, both in cases where subjective knowledge is available or not Press (2009). Kass and Wasserman (1996) propose different ways of defining the level of non-informativeness of a prior distribution. The most famous method is the Maximum Entropy (ME) prior distribution that has been popularized by Jaynes (1957). In a Bayesian context, ME and Maximal Data Information (MDI) priors have been studied by Zellner (1996), Soofi (2000). Other candidates for objective priors are the so-called matching priors Reid, Mukerjee, and Fraser (2003), which are priors such that the Bayesian posterior credible intervals correspond to confidence intervals of the sampling model. Moreover, when a simpler model is known, the Penalizing Complexity (PC) priors are yet another rationale of choosing an objective (or reference) prior distribution Simpson et al. (2017).

In this paper, we will focus on the reference prior theory. First introduced in Bernardo (1979a) and further formalized in Berger, Bernardo, and Sun (2009), the main rationale behind the reference prior theory is the maximization of the information brought by the data during Bayesian inference. Specifically, reference priors (RPs) are constructed to maximize the mutual information metric, which is defined as a divergence between itself and the posterior. In this way, it ensures that the data plays a dominant role in the Bayesian framework. This approach has been extensively studied (see e.g. Bernardo (1979b), Clarke and Barron (1994), Van Biesbroeck (2024a)) and applied to various statistical models, such as Gaussian process-based models Paulo (2005), Gu and Berger (2016), generalized linear models Natarajan and Kass (2000), and even Bayesian Neural Networks Gao, Ramesh, and Chaudhari (2022). The RPs are recognized for their objective nature in practical studies D'Andrea (2021), Li and Gu (2021), Van Biesbroeck et al. (2024), yet they suffer from their low computational feasibility. Indeed, the expression of the RPs often leads to an intricate theoretical expression, which necessitates a heavy numerical cost to be derived that becomes even more cumbersome as the dimensionality of the problem increases. Moreover, in many applications, a posteriori estimates are obtained using Markov Chain Monte Carlo (MCMC) methods, which require a large number of prior evaluations, further compounding the computational burden.

In general, when we look for sampling or approximating a probability distribution, several approaches arise and may be used within a Bayesian framework. In this work, we focus on variational inference methods. Variational inference seeks to approximate a complex target distribution p, —e.g. a posterior— by optimizing over a family of simpler parameterized distributions q_λ. The goal then is to find the distribution q_{λ^*} that is the best approximation of p by minimizing a divergence, such as the Kullback-Leibler (KL) divergence. Variational inference methods have been widely adopted in various contexts, including popular models such as Variational Autoencoders (VAEs) Kingma and Welling (2019), which are a class of generative models where one wants to learn the underlying distribution of data samples. We can also mention normalizing flows Papamakarios et al. (2021), Kobyzev, Prince, and Brubaker (2021), which consider diffeomorphism transformations to recover the density of the approximated distribution from the simpler one taken as input.

When it resorts to approximate RPs, it is possible to leverage the optimal characteristic of the

reference prior (that is, it maximizes the mutual information metric) instead of directly maximizing a divergence between a target and an output. Indeed, the mutual information metric does not depend on the target distribution that we want to reach so iterative derivations of the theoretical RP are not necessary. In Nalisnick and Smyth (2017), the authors propose a variational inference procedure to approximate the RP using a lower bound of the mutual information as an optimization criterion. In Gauchy et al. (2023), a variational inference procedure is proposed using stochastic gradient descent of the mutual information criterion and illustrated on simple statistical models.

By building on these foundations, this paper proposes a novel variational inference algorithm to compute RPs. As in Nalisnick and Smyth (2017) and Gauchy et al. (2023), the RP is approximated in a parametric family of probability distributions implicitly defined by the push-forward probability distribution through a nonlinear function (see e.g. Papamakarios et al. (2021) and Marzouk et al. (2016)). We will focus in this paper to push-forward probability measures through neural networks. In comparison with the previous works, we benchmark extensively our algorithm on statistical models of different complexity and nature to ensure its robustness. We also extend our algorithm to handle a more general definition of RPs Van Biesbroeck (2024a), where a generalized mutual information criterion is defined using f-divergences. In this paper, we restrict the different benchmarks to α-divergences. Additionally, we extend the framework to allow the integration of linear constraints on the prior in the pipeline. That last feature permits handling situations where the RP may be improper (i.e. it integrates to infinity). Improper priors pose a challenge because (i) one can not sample from the a priori distribution, and (ii) they do not ensure that the posterior is proper, jeopardizing a posteriori inference. Recent work detailed in Van Biesbroeck (2024b) introduces linear constraints that ensure the proper aspects of RPs. Our algorithm incorporates these constraints, providing a principled way to address improper priors and ensuring that the resulting posterior distributions are well-defined and suitable for practical use.

First, we will introduce the reference prior theory of Bernardo (1979b) and the recent developments around generalized reference priors made by Van Biesbroeck (2024a) in Section 2. Next, the variational approximation of the reference prior (VA-RP) methodology is detailed in Section 3. A stochastic gradient algorithm is proposed, as well as an augmented Lagrangian algorithm for the constrained optimization problem, for learning the parameters of an implicitly defined probability density function that will approximate the reference prior. Moreover, a mindful trick to sample from the posterior distribution by MCMC using the implicitly defined prior distribution is proposed. In Section 4, different numerical experiments from various test cases are carried out in order to benchmark the VA-RP. Analytical statistical models where the true asymptotic RP is known are tested to allow comparison between the VA-RP and the true asymptotic RP.

2 Reference priors theory

The reference prior theory fits into the usual framework of statistical inference. The situation is the following: we observe i.i.d data samples $\mathbf{X} = (X_1, ..., X_N) \in \mathcal{X}^N$ with $\mathcal{X} \subset \mathbb{R}^d$. We suppose that the likelihood function $L_N(\mathbf{X}\,|\,\theta) = \prod_{i=1}^N L(X_i\,|\,\theta)$ is known and $\theta \in \Theta \subset \mathbb{R}^q$ is the parameter we want to infer. Since we use the Bayesian framework, θ is considered to be a random variable with a prior distribution π. We also define the marginal likelihood $p_{\pi,N}(\mathbf{X}) = \int_\Theta \pi(\theta) L_N(\mathbf{X}\,|\,\theta) d\theta$ associated to the marginal probability measure $\mathbb{P}_{\pi,N}$. The non-asymptotic RP, first introduced in Bernardo (1979a) and formalized in Berger, Bernardo, and Sun (2009), is defined to be one of the priors verifying:

$$\pi^* \in \underset{\pi \in \mathscr{P}}{\operatorname{argmax}}\, I(\pi; L_N)\,, \tag{1}$$

where $\mathscr{P}$ is a class of admissible probability distributions and $I(\pi; L_N)$ is the mutual information for the prior π and the likelihood L_N between the random variable of the parameters $\theta \sim \pi$ and the random variable of the data $\mathbf{X} \sim \mathbb{P}_{\pi,N}$:

$$I(\pi; L_N) = \int_{\mathcal{X}^N} \mathrm{KL}(\pi(\cdot \,|\, \mathbf{X}) \,\|\, \pi) p_{\pi,N}(\mathbf{X}) d\mathbf{X} \tag{2}$$

Hence, π^* is a prior that maximizes the Kullback-Leibler divergence between itself and its posterior averaged by the marginal distribution of datasets. The Kullback-Leibler divergence between two probability measures of density p and q defined on a generic set Ω writes:

$$\mathrm{KL}(p \,\|\, q) = \int_{\Omega} \log\left(\frac{p(\omega)}{q(\omega)}\right) p(\omega) d\omega.$$

Thus, π^* is the prior that maximises the influence of the data on the posterior distribution, justifying its reference (or objective) properties. The RP π^* can also be interpreted using channel coding and information theory MacKay (2003) (chapter 9). Indeed, remark that $I(\pi; L_N)$ corresponds to the mutual information $I(\theta, \mathbf{X})$ between the random variable $\theta \sim \pi$ and the data $\mathbf{X} \sim \mathbb{P}_{\pi,N}$, it measures the information that conveys the data $\mathbf{X}$ about the parameters θ. The maximal value of this mutual information is defined as the channel's capacity. The RP thus corresponds to the prior distribution that maximizes the information about θ conveyed by the data $\mathbf{X}$.

Using Fubini's theorem and Bayes' theorem, we can derive an alternative and more practical expression for the mutual information :

$$I(\pi; L_N) = \int_{\Theta} \mathrm{KL}(L_N(\cdot \,|\, \theta) \| p_{\pi,N}) \pi(\theta) d\theta. \tag{3}$$

A more generalized definition of RPs has been proposed in Van Biesbroeck (2024a) using f-divergences. The f-divergence mutual information is defined by

$$I_{\mathrm{D}_f}(\pi; L_N) = \int_{\Theta} \mathrm{D}_f(p_{\pi,N} \| L_N(\cdot \,|\, \theta)) \pi(\theta) d\theta \,, \tag{4}$$

with

$$\mathrm{D}_f(p \,\|\, q) = \int_{\Omega} f\left(\frac{p(\omega)}{q(\omega)}\right) q(\omega) d\omega. \,,$$

where f is usually chosen to be a convex function mapping 1 to 0. Remark that the classical mutual information is obtained by choosing $f = -\log$, indeed, $\mathrm{D}_{-\log}(p \,\|\, q) = \mathrm{KL}(q \,\|\, p)$. The formal RP is defined as N goes to infinity, but since we want to develop an algorithm to approximate the distribution of the RP, we are restricted to the case where N takes a finite value. However, the limit case $N \to +\infty$ is relevant because it has been shown in Clarke and Barron (1994), Van Biesbroeck (2024a) that the solution of this asymptotic problem is the Jeffreys prior when the mutual information is expressed as in Equation 2, or when it is defined using an α-divergence, as in Equation 4 with $f = f_\alpha$, where:

$$f_\alpha(x) = \frac{x^\alpha - \alpha x - (1 - \alpha)}{\alpha(\alpha - 1)}, \quad \alpha \in (0, 1). \tag{5}$$

The Jeffreys prior, denoted by J, is defined as follows:

$$J(\theta) \propto \det(\mathcal{I}(\theta))^{1/2} \quad \text{with} \quad \mathcal{I}(\theta) = -\int_{\mathcal{X}^N} \frac{\partial^2 \log L_N}{\partial \theta^2}(\mathbf{X} \,|\, \theta) \cdot L_N(\mathbf{X} \,|\, \theta) \, d\mathbf{X}.$$

We suppose that the likelihood function is smooth such that the Fisher information matrix $\mathcal{I}$ is well-defined. The Jeffreys prior and the RP have the relevant property to be "invariant by reparametrization' ':

$$\forall \varphi \text{ diffeomorphism}, \quad J(\theta) = \left|\frac{\partial \varphi}{\partial \theta}\right| \cdot J(\varphi(\theta)).$$

This property expresses non-information in the sense that if there is no information on θ, there should not be more information on $\varphi(\theta)$ when φ is a diffeomorphism: an invertible and differentiable transformation.

Actually, the historical definition of RPs involves the KL-divergence in the definition of the mutual information. Yet the use of α-divergences instead is relevant because they can be seen as a continuous path between the KL-divergence and the Reverse-KL-divergence when α varies from 0 to 1. We can also mention that for $\alpha = 1/2$, the α-divergence is the squared Hellinger distance whose square root is a metric since it is symmetric and verifies the triangle inequality.

Technically, the formal RP is constructed such that its projection on every compact subset (or open subset in Muré (2018)) of Θ maximizes asymptotically the mutual information, which allows for improper distributions to be RPs in some cases. The Jeffreys prior is itself often improper.

In our algorithm we consider probability distributions defined on the space Θ and not on sequences of subsets. A consequence of this statement is that our algorithm may tend to approximate improper priors in some cases. Indeed, any given sample by our algorithm results, by construction, from a proper distribution, which is expected to be a good approximation of the solution of the optimization problem expressed in Equation 1. If N is large enough, the latter should be close to the —potentially improper— theoretical RP. This approach is justified to some extent since in the context of Q-vague convergence defined in Bioche and Druilhet (2016) for instance, improper priors can be the limit of sequences of proper priors. Although this theoretical notion of convergence is defined, no concrete metric is given, making quantification of the difference between proper and improper priors infeasible in practice. Furthermore, as mentioned in the introduction, improper priors can also compromise the validity of {a posteriori} estimates in some cases. To address this issue, we adapted our algorithm to handle the developments made in Van Biesbroeck (2024b), which suggest a method to define proper RPs by simply resolving a constrained version of the initial optimization problem:

$$\tilde{\pi}^* \in \underset{\substack{\pi \text{prior} \\ \text{s.t. } \mathscr{C}(\pi)<\infty}}{\operatorname{argmax}} I_{\mathrm{D}_{f_\alpha}}(\pi; L_N), \tag{6}$$

where $\mathscr{C}(\pi)$ defines a constraint of the form $\int_\Theta a(\theta)\pi(\theta)d\theta$, a being a positive function. When the mutual information in the above optimization problem is defined from an α-divergence, and when a verifies

$$\int_\Theta J(\theta)a(\theta)^{1/\alpha}d\theta < \infty \quad \text{and} \quad \int_\Theta J(\theta)a(\theta)^{1+1/\alpha}d\theta < \infty, \tag{7}$$

the author has proven that the constrained RP $\tilde{\pi}^*$ asymptotically takes the following form:

$$\tilde{\pi}^*(\theta) \propto J(\theta)a(\theta)^{1/\alpha},$$

which is proper.

3 Variational approximation of the reference prior (VA-RP)

3.1 Implicitly defined parametric probability distributions using neural networks

Variational inference refers to techniques that aim to approximate a probability distribution by solving an optimization problem —that often takes a variational form, such as maximizing evidence lower

bound (ELBO) Kingma and Welling (2014). It is thus relevant to consider them for approximating RPs, as the goal is to maximize, w.r.t. the prior, the mutual information defined in Equation 3.

We restrict the set of priors to a parametric space $\{\pi_\lambda,\ \lambda \in \Lambda\}$, $\Lambda \subset \mathbb{R}^L$, reducing the original optimization problem into a finite-dimensional one. The optimization problem in Equation 1 or Equation 6 becomes finding $\underset{\lambda\in\Lambda}{\operatorname{argmax}}\, I_{\mathrm{D}_f}(\pi_\lambda; L_N)$. Our approach is to define the set of priors π_λ implicitly, as in Gauchy et al. (2023):

$$\theta \sim \pi_\lambda \iff \theta = g(\lambda, \varepsilon) \quad \text{and} \quad \varepsilon \sim \mathbb{P}_\varepsilon.$$

Here, g is a measurable function parameterized by λ, typically a neural network with λ corresponding to its weights and biases, and we impose that g is differentiable with respect to λ. The variable ε can be seen as a latent variable. It has an easy-to-sample distribution $\mathbb{P}_\varepsilon$ with a simple density function. In practice we use the centered multivariate Gaussian $\mathcal{N}(0, \mathbb{I}_{p\times p})$. The construction described above allows the consideration of a vast family of priors. However, except in very simple cases, the density of π_λ is not known and cannot be evaluated. Only samples of $\theta \sim \pi_\lambda$ can be obtained.

In the work of Nalisnick and Smyth (2017), this implicit sampling method is compared to several other algorithms used to learn RPs in the case of one-dimensional models. Among these methods, we can mention an algorithm proposed by Berger, Bernardo, and Sun (2009) which does not sample from the RP but only evaluates it for specific points, or an MCMC-based approach by Lafferty and Wasserman (2001), which is inspired from the previous one but can sample from the RP.

According to this comparison, implicit sampling is, in the worst case, competitive with the other methods, but achieves state-of-the-art results in the best case. Hence, computing the variational approximation of the RP, which we will refer to as the VA-RP, seems to be a promising technique.

The situations presented by Gauchy et al. (2023) and Nalisnick and Smyth (2017) are in dimension one and use the Kullback-Leibler divergence within the definition of the mutual information.

The construction of the algorithm that we propose in the following accommodates multi-dimensional modeling. It is also compatible with the extended form of the mutual information, as defined in Equation 3 from an f-divergence.

The choice of the neural network is up to the user, we will showcase in our numerical applications simple networks, composed of one fully connected linear layer and one activation function. However, the method can be used with deeper networks, such as normalizing flows Papamakarios et al. (2021), or larger networks obtained through a mixture model of smaller networks utilizing the "Gumbel-Softmax trick' ' Jang, Gu, and Poole (2017) for example. Such choices lead to more flexible parametric distributions, but increase the difficulty of fine-tuning hyperparameters.

3.2 Learning the VA-RP using stochastic gradient algorithm

The VA-RP is formulated as the solution to the following optimization problem:

$$\pi_{\lambda^*} = \underset{\lambda\in\Lambda}{\operatorname{argmax}}\, \mathcal{O}_{\mathrm{D}_f}(\pi; L_N), \tag{8}$$

where π_λ is parameterized through the relation between a latent variable ε and the parameter θ, as outlined in the preceding Section. The function $\mathcal{O}_{\mathrm{D}_f}$ is called the objective function, it is maximized using stochastic gradient optimization, following the approach described in Gauchy et al. (2023).

It is intuitive to fix $\mathcal{O}_{\mathrm{D}_f}$ to equal I_{D_f}, in order to maximize the mutual information of interest. In this Section, we suggest alternative objective functions that can be considered to compute the VA-RP.

Our method is adaptable to any objective function $\mathcal{O}_{\mathrm{D}_f}$ admitting a gradient w.r.t. $\lambda = (\lambda_1, \dots, \lambda_L)$ that takes the form

$$\frac{\partial \mathcal{O}_{\mathrm{D}_f}}{\partial \lambda_l}(\pi_\lambda; L_N) = \mathbb{E}_\varepsilon \left[\sum_{j=1}^{q} \frac{\partial \tilde{\mathcal{O}}_{\mathrm{D}_f}}{\partial \theta_j}(g(\lambda, \varepsilon)) \frac{\partial g_j}{\partial \lambda_l}(\lambda, \varepsilon) \right] \tag{9}$$

for any $l \in \{1, \dots, L\}$, where $\tilde{\mathcal{O}}_{\mathrm{D}_f}$ is independent of λ. This framework allows for flexible implementation, as it permits the separation of sampling and differentiation operations:

- The gradient of $\tilde{\mathcal{O}}_{\mathrm{D}_f}$ mostly relies on random sampling and depends only on the likelihood L_N and the function f.
- The gradient of g is computed independently. In practice, it is possible to leverage usual differentiation techniques for the neural network. In our work, we rely on PyTorch's automatic differentiation feature "autograd' ' Paszke et al. (2019).

This separation is advantageous as automatic differentiation tools —such as autograd— are well-suited to differentiating complex networks but struggle with functions incorporating randomness.

This way, the optimization problem can be addressed using stochastic gradient optimization, approximating at each step the gradient in Equation 9 via Monte Carlo estimates. In our experiments, the implementation of the algorithm is done with the popular Adam optimizer (Kingma and Ba (2015)), with its default hyperparameters, $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The learning rate is tuned more specifically for each numerical benchmark.

Concerning the choice of objective function, we verify that I_{D_f} is compatible with our method by computing its gradient:

$$\begin{aligned} \frac{\partial I_{\mathrm{D}_f}}{\partial \lambda_l}(\pi_\lambda; L_N) = \mathbb{E}_\varepsilon \left[\sum_{j=1}^{q} \frac{\partial \tilde{I}}{\partial \theta_j}(g(\lambda, \varepsilon)) \frac{\partial g_j}{\partial \lambda_l}(\lambda, \varepsilon) \right] \\ + \mathbb{E}_{\theta \sim \pi_\lambda} \left[\mathbb{E}_{\mathbf{X} \sim L_N(\cdot|\theta)} \left[\frac{1}{L_N(\mathbf{X}\,|\,\theta)} \frac{\partial p_\lambda}{\partial \lambda_l}(\mathbf{X}) f' \left(\frac{p_\lambda(\mathbf{X})}{L_N(\mathbf{X}\,|\,\theta)} \right) \right] \right], \end{aligned}$$

where:

$$\frac{\partial \tilde{I}}{\partial \theta_j}(\theta) = \mathbb{E}_{\mathbf{X} \sim L_N(\cdot\,|\,\theta)} \left[\frac{\partial \log L_N}{\partial \theta_j}(\mathbf{X}\,|\,\theta) F \left(\frac{p_\lambda(\mathbf{X})}{L_N(\mathbf{X}\,|\,\theta)} \right) \right],$$

with $F(x) = f(x) - x f'(x)$ and p_λ is a shortcut notation for $p_{\pi_\lambda, N}$ being the marginal distribution under π_λ. The details are given in the Section 6.1. Remark that only the case $f = -\log$ is considered by Gauchy et al. (2023), but it leads to a simplification of the gradient since the second term vanishes. Each term in the above equations is approximated as follows:

$$\begin{cases} p_\lambda(\mathbf{X}) = \mathbb{E}_{\theta \sim \pi_\lambda}[L_N(\mathbf{X}\,|\,\theta)] \approx \dfrac{1}{T} \displaystyle\sum_{t=1}^{T} L_N(\mathbf{X}\,|\,g(\lambda, \varepsilon_t)) \quad \text{where} \quad \varepsilon_1, \dots, \varepsilon_T \sim \mathbb{P}_\varepsilon \\ \dfrac{\partial \tilde{I}}{\partial \theta_j}(\theta) \approx \dfrac{1}{U} \displaystyle\sum_{u=1}^{U} \dfrac{\partial \log L_N}{\partial \theta_j}(\mathbf{X}^u\,|\,\theta) F \left(\dfrac{p_\lambda(\mathbf{X}^u)}{L_N(\mathbf{X}^u\,|\,\theta)} \right) \quad \text{where} \quad \mathbf{X}^1, \dots, \mathbf{X}^U \sim \mathbb{P}_{\mathbf{X}|\theta}. \end{cases} \tag{10}$$

In their work, Nalisnick and Smyth (2017) propose an alternative objective function to optimize, that we call B_{D_f}.

This function corresponds to a lower bound of the mutual information. It is derived from an upper bound on the marginal distribution and relies on maximizing the likelihood. Their approach is only presented for $f = -\log$, we generalize the lower bound for any decreasing function f:

$$B_{\mathrm{D}_f}(\pi; L_N) = \int_\Theta \int_{\mathcal{X}^N} f \left(\frac{L_N(\mathbf{X}\,|\,\hat{\theta}_{MLE})}{L_N(\mathbf{X}\,|\,\theta)} \right) \pi(\theta) L_N(\mathbf{X}\,|\,\theta) d\mathbf{X} d\theta, \tag{11}$$

where $\hat{\theta}_{MLE}$ being the maximum likelihood estimator (MLE). It only depends on the likelihood and not on λ which simplifies the gradient computation:

$$\frac{\partial B_{\mathrm{D}_f}}{\partial \lambda_l}(\pi_\lambda; L_N) = \mathbb{E}_\varepsilon \left[\sum_{j=1}^{q} \frac{\partial \tilde{B}}{\partial \theta_j}(g(\lambda, \varepsilon)) \frac{\partial g_j}{\partial \lambda_l}(\lambda, \varepsilon) \right],$$

where:

$$\frac{\partial \tilde{B}}{\partial \theta_j}(\theta) = \mathbb{E}_{\mathbf{X} \sim L_N(\cdot \,|\, \theta)} \left[\frac{\partial \log L_N}{\partial \theta_j}(\mathbf{X} \,|\, \theta) F\left(\frac{L_N(\mathbf{X} \,|\, \hat{\theta}_{MLE})}{L_N(\mathbf{X} \,|\, \theta)} \right) \right].$$

Its form corresponds to the one expressed in Equation 9.

Given that $p_\lambda(\mathbf{X}) \leq \max_{\theta' \in \Theta} L_N(\mathbf{X} \,|\, \theta') = L_N(\mathbf{X} \,|\, \hat{\theta}_{MLE})$ for all λ, we have $B_{\mathrm{D}_f}(\pi_\lambda; L_N) \leq I_{\mathrm{D}_f}(\pi_\lambda; L_N)$.

Since f_α, used in α-divergence (Equation 5), is not decreasing, we replace it with $\hat{f}_\alpha$ defined hereafter, because $\mathrm{D}_{f_\alpha} = \mathrm{D}_{\hat{f}_\alpha}$:

$$\hat{f}_\alpha(x) = \frac{x^\alpha - 1}{\alpha(\alpha - 1)} = f_\alpha(x) + \frac{1}{\alpha - 1}(x - 1).$$

The use of this function results in a more stable computation overall. Moreover, one argument for the use of α-divergences rather that the KL-divergence, is that we have an universal and explicit upper bound on the mutual information:

$$I_{\mathrm{D}_{f_\alpha}}(\pi; L_N) = I_{\mathrm{D}_{\hat{f}_\alpha}}(\pi; L_N) \leq \hat{f}_\alpha(0) = \frac{1}{\alpha(1 - \alpha)}.$$

This bound can be an indicator on how well the mutual information is optimized, although there is no guarantee that it can be attained in general.

The gradient of the objective function B_{D_f} can be approximated via Monte Carlo, in the same manner as in Equation 10.

It requires to compute the MLE, which can also be done using samples of ε:

$$L_N(\mathbf{X} \,|\, \hat{\theta}_{MLE}) \approx \max_{t \in \{1, \ldots, T\}} L_N(\mathbf{X} \,|\, g(\lambda, \varepsilon_t)) \quad \text{where} \quad \varepsilon_1, \ldots, \varepsilon_T \sim \mathbb{P}_\varepsilon.$$

3.3 Adaptation for the constrained VA-RP

Reference priors are often criticized, because it can lead to improper posteriors. However, the variational optimization problem defined in Equation 8 can be adapted to incorporate simple constraints on the prior. As mentioned in Section 2, there exist specific constraints that would make the theoretical solution proper.

This is also a way to incorporate expert knowledge to some extent. We consider K constraints of the form:

$$\forall\, k \in \{1, \ldots, K\}, \;\; \mathscr{C}_k(\pi_\lambda) = \mathbb{E}_{\theta \sim \pi_\lambda}\left[a_k(\theta)\right] - b_k,$$

with $a_k : \Theta \mapsto \mathbb{R}^+$ integrable and linearly independent functions, and $b_k \in \mathbb{R}$. We then adapt the optimization problem in Equation 8 to propose the following constrained optimization problem:

$$\begin{aligned} &\pi^C_{\lambda^*} \in \operatorname*{argmax}_{\lambda \in \Lambda} \mathscr{O}_{\mathrm{D}_f}(\pi_\lambda; L_N) \\ &\text{subject to} \quad \forall\, k \in \{1, \ldots, K\}, \;\; \mathscr{C}_k(\pi_\lambda) = 0, \end{aligned}$$

where $\pi^C_{\lambda^*}$ is the constrained VA-RP. The optimization problem with the mutual information has an explicit asymptotic solution for proper priors verifying the previous conditions:

- In the case of the KL-divergence (Bernardo (2005)):

$$\pi^C(\theta) \propto J(\theta) \exp\left(1 + \sum_{k=1}^{K} \nu_k a_k(\theta)\right).$$

- In the case of α-divergences (Van Biesbroeck (2024b)):

$$\pi^C(\theta) \propto J(\theta) \left(1 + \sum_{k=1}^{K} \nu_k a_k(\theta)\right)^{1/\alpha}.$$

where $\nu_1, \ldots, \nu_K \in \mathbb{R}$ are constants determined by the constraints.

Recent work by Van Biesbroeck (2024b) makes it possible to build a proper reference prior under a relevant constraint function with α-divergence. The theorem considers $a : \Theta \mapsto \mathbb{R}^+$ which verifies the conditions expressed in Equation 7. Letting $\mathscr{P}_a$ be the set of priors π on Θ such that $\pi \cdot a \in L^1$, the reference prior $\tilde{\pi}^*$ under the constraint $\tilde{\pi}^* \in \mathscr{P}_a$ is:

$$\tilde{\pi}^*(\theta) \propto J(\theta) a(\theta)^{1/\alpha}.$$

We propose the following general method to approximate the VA-RP under such constraints:

- Compute the VA-RP $\pi_\lambda \approx J$, in the same manner as for the unconstrained case.
- Estimate the constants $\mathscr{K}$ and c using Monte Carlo samples from the VA-RP, as:

$$\mathscr{K}_\lambda = \int_\Theta \pi_\lambda(\theta) a(\theta)^{1/\alpha} d\theta \approx \int_\Theta J(\theta) a(\theta)^{1/\alpha} d\theta = \mathscr{K},$$

$$c_\lambda = \int_\Theta \pi_\lambda(\theta) a(\theta)^{1+(1/\alpha)} d\theta \approx \int_\Theta J(\theta) a(\theta)^{1+(1/\alpha)} d\theta = c.$$

- Since we have the equality:

$$\mathbb{E}_{\theta \sim \tilde{\pi}^*}[a(\theta)] = \int_\Theta \tilde{\pi}^*(\theta) a(\theta) d\theta = \frac{1}{\mathscr{K}} \int_\Theta J(\theta) a(\theta)^{1+(1/\alpha)} d\theta = \frac{c}{\mathscr{K}},$$

we compute the constrained VA-RP using the constraint : $\mathbb{E}_{\theta \sim \pi_{\lambda'}}[a(\theta)] = c_\lambda / \mathscr{K}_\lambda$ to approximate $\pi_{\lambda'} \approx \tilde{\pi}^*$.

One might use different variational approximations for π_λ and $\pi_{\lambda'}$ because J and $\tilde{\pi}^*$ could have very different forms depending on the function a.

The idea is to solve the constrained optimization problem as an unconstrained problem but with a Lagrangian as the objective function. We take the work of Nocedal and Wright (2006) as support.

We denote η the Lagrange multiplier. Instead of using the usual Lagrangian function, Nocedal and Wright (2006) suggest adding a term defined with $\tilde{\eta}$, a vector with positive components which serve as penalization coefficients, and η' which can be thought of a prior estimate of η, although not in a Bayesian sense. The objective is to find a saddle point (λ^*, η^*) which is a solution of the updated optimization problem:

$$\max_\lambda \left(\min_\eta \mathscr{O}_{\mathrm{D}_f}(\pi_\lambda; L_N) + \sum_{k=1}^{K} \eta_k \mathscr{C}_k(\pi_\lambda) + \sum_{k=1}^{K} \frac{1}{2\tilde{\eta}_k} (\eta_k - \eta'_k)^2 \right).$$

One can see that the third term serves as a penalization for large deviations from η'. The minimization on η is feasible because it is a convex quadratic, and we get $\eta = \eta' - \tilde{\eta} \cdot \mathscr{C}(\pi_\lambda)$. Replacing η by its expression leads to the resolution of the problem:

$$\max_{\lambda} \mathscr{O}_{\mathrm{D}_f}(\pi_\lambda; L_N) + \sum_{k=1}^{K} \eta'_k \mathscr{C}_k(\pi_\lambda) - \sum_{k=1}^{K} \frac{\tilde{\eta}_k}{2} \mathscr{C}_k(\pi_\lambda)^2.$$

This motivates the definition of the augmented Lagrangian:

$$\mathscr{L}_A(\lambda, \eta, \tilde{\eta}) = \mathscr{O}_{\mathrm{D}_f}(\pi_\lambda; L_N) + \sum_{k=1}^{K} \eta_k \mathscr{C}_k(\pi_\lambda) - \sum_{k=1}^{K} \frac{\tilde{\eta}_k}{2} \mathscr{C}_k(\pi_\lambda)^2.$$

Its gradient has a form that which is compatible with our algorithm, as depicted in Section 3.2 (see Equation 9):

$$\begin{aligned}\frac{\partial \mathscr{L}_A}{\partial \lambda}(\lambda, \eta, \tilde{\eta}) &= \frac{\partial \mathscr{O}_{\mathrm{D}_f}}{\partial \lambda}(\pi_\lambda; L_N) + \mathbb{E}_\varepsilon \left[\left(\sum_{k=1}^{K} \frac{\partial a_k}{\partial \theta}(g(\lambda, \varepsilon))(\eta_k - \tilde{\eta}_k \mathscr{C}_k(\pi_\lambda)) \right) \frac{\partial g}{\partial \lambda}(\lambda, \varepsilon) \right] \\ &= \mathbb{E}_\varepsilon \left[\left(\frac{\partial \tilde{\mathscr{O}}}{\partial \theta}(g(\lambda, \varepsilon)) + \sum_{k=1}^{K} \frac{\partial a_k}{\partial \theta}(g(\lambda, \varepsilon))(\eta_k - \tilde{\eta}_k \mathscr{C}_k(\pi_\lambda)) \right) \frac{\partial g}{\partial \lambda}(\lambda, \varepsilon) \right].\end{aligned}$$

In practice, the augmented Lagrangian algorithm is of the form:

$$\begin{cases} \lambda^{t+1} = \operatorname*{argmax}_{\lambda} \mathscr{L}_A(\lambda, \eta^t, \tilde{\eta}) \\ \forall k \in \{1, \dots, K\},\ \eta_k^{t+1} = \eta_k^t - \tilde{\eta}_k \cdot \mathscr{C}_k(\pi_{\lambda^{t+1}}). \end{cases}$$

In our implementation, η is updated every 100 epochs. The penalty parameter $\tilde{\eta}$ can be interpreted as the learning rate of η, we use an adaptive scheme inspired by Basir and Senocak (2023) where we check if the largest constraint value $\|\mathscr{C}(\pi_\lambda)\|_\infty$ is higher than a specified threshold M or not. If $\|\mathscr{C}(\pi_\lambda)\|_\infty > M$, we multiply $\tilde{\eta}$ by ν, otherwise we divide by ν. We also impose a maximum value $\tilde{\eta}_{max}$.

3.4 Posterior sampling using implicitly defined prior distributions

Although our main object of study is the prior distribution, one needs to find the posterior distribution given an observed dataset $\mathbf{X}$ in order to do the inference on θ. The posterior is of the form :

$$\pi_\lambda(\theta \,|\, \mathbf{X}) = \frac{\pi_\lambda(\theta) L_N(\mathbf{X} \,|\, \theta)}{p_\lambda(\mathbf{X})}.$$

As discussed in the introduction, one can approximate the posterior distribution when knowing the prior either by using MCMC or variational inference. In both cases, knowing the marginal distribution is not required. Indeed, MCMC samplers inspired by the Metropolis-Hastings algorithm can be applied, even if the posterior distribution is only known up to a multiplicative constant. The same can be said for variational approximation since the ELBO can be expressed without the marginal.

The issue here is that the density function $\pi_\lambda(\theta)$ is not explicit and can not be evaluated, except for very simple cases. However, we imposed that the distribution of the variable ε is simple enough so one is able to evaluate its density. We propose to use ε as the variable of interest instead of θ because it lets us circumvent this issue. In practice, the idea is to reverse the order of operations : instead of sampling ε, then transforming ε into θ, which defines the prior on θ, and finally sampling posterior samples of θ given X, one can proceed as follows :

- Define the posterior distribution on ε :

$$\pi_{\varepsilon,\lambda}(\varepsilon \,|\, \mathbf{X}) = \frac{p_\varepsilon(\varepsilon) L_N(\mathbf{X} \,|\, g(\lambda, \varepsilon))}{p_\lambda(\mathbf{X})} \,,$$

  where p_ε is the probability density function of ε. $\pi_{\varepsilon,\lambda}(\varepsilon \,|\, \mathbf{X})$ is known up to a multiplicative constant since the marginal p_λ is intractable in general. It is indeed a probability distribution on $\mathbb{R}^p$ because :

$$p_\lambda(\mathbf{X}) = \int_\Theta \pi_\lambda(\theta) L_N(\mathbf{X} \,|\, \theta) d\theta = \int_{\mathbb{R}^p} L_N(\mathbf{X} \,|\, g(\lambda, \varepsilon)) d\mathbb{P}_\varepsilon$$

- Sample posterior ε samples from the previous distribution, approximated by MCMC or variational inference.
- Apply the transformation $\theta = g(\lambda, \varepsilon)$, and one gets posterior θ samples : $\theta \sim \pi_\lambda(\cdot \,|\, \mathbf{X})$.

More precisely, we denote for a fixed dataset $\mathbf{X}$:

$$\theta \sim \tilde{\pi}_\lambda(\cdot \,|\, \mathbf{X}) \iff \theta = g(\lambda, \varepsilon) \quad \text{with} \quad \varepsilon \sim \pi_{\varepsilon,\lambda}(\cdot \,|\, \mathbf{X}).$$

The previous approach is valid because $\pi_\lambda(\cdot \,|\, \mathbf{X})$ and $\tilde{\pi}_\lambda(\cdot \,|\, \mathbf{X})$ lead to the same distribution, as proven by the following derivation : let φ be a bounded and measurable function on Θ.

Using the definitions of the different distributions, we have that:

$$\begin{aligned}
\int_\Theta \varphi(\theta) \tilde{\pi}_\lambda(\theta \,|\, \mathbf{X}) d\theta &= \int_{\mathbb{R}^p} \varphi(g(\lambda, \varepsilon)) \pi_{\varepsilon,\lambda}(\varepsilon \,|\, \mathbf{X}) d\varepsilon \\
&= \int_{\mathbb{R}^p} \varphi(g(\lambda, \varepsilon)) \frac{p_\varepsilon(\varepsilon) L_N(X \,|\, g(\lambda, \varepsilon))}{p_\lambda(\mathbf{X})} d\varepsilon \\
&= \int_\Theta \varphi(\theta) \pi_\lambda(\theta) \frac{L_N(\mathbf{X} \,|\, \theta)}{p_\lambda(\mathbf{X})} d\theta \\
&= \int_\Theta \varphi(\theta) \pi_\lambda(\theta \,|\, \mathbf{X}) d\theta.
\end{aligned}$$

As mentioned in the last Section, when the RP is improper, we compare the posterior distributions, namely, the exact reference posterior when available and the posterior obtained from the VA-RP using the previous method. Altogether, we are able to sample θ from the posterior even if the density of the parametric prior π_λ on θ is unavailable due to an implicit definition of the prior distribution.

For our computations, we choose MCMC sampling, namely an adaptive Metropolis-Hastings sampler with a multivariate Gaussian as the proposition distribution. The adaptation scheme is the following: for each batch of iterations, we monitor the acceptance rate and we adapt the variance parameter of the Gaussian proposition in order to have an acceptance rate close to 40%, which is the advised value Gelman et al. (2013) for models in small dimensions. We refer to this algorithm as MH(ε). Because we apply MCMC sampling on variable $\varepsilon \in \mathbb{R}^p$ with a reasonable value for p, we expect this step of the algorithm to be fast compared to the computation of the VA-RP.

One could also use classic variational inference on ε instead, but the parametric set of distributions must be chosen wisely. In VAEs for instance, multivariate Gaussian are often considered since it simplifies the KL-divergence term in the ELBO. However, this might be too simplistic in our case since we must apply the neural network g to recover θ samples. This means that the approximated posterior on θ belongs to a very similar set of distributions to those used for the VA-RP, since we already used a multivariate Gaussian for the prior on ε. On the other hand, applying once again the implicit sampling approach does not exploit the additional information we have on $\pi_{\varepsilon,\lambda}(\varepsilon \,|\, \mathbf{X})$ compared to $\pi_\lambda(\theta)$, specifically, that its density function is known up to a multiplicative constant. Hence, we argue that using a Metropolis-Hastings sampler is more straightforward in this situation.

4 Numerical experiments

We want to apply our algorithm to different statistical models, the first one is the multinomial model, which is the simplest in the sense that the target distributions —the Jeffreys prior and posterior— have explicit expressions and are part of a usual parametric family of proper distributions. The second model —the probit model— will be highlighted with supplementary computations, in regards to the assessment of the stability of our stochastic algorithm, and also with the addition of a moment constraint.

The one-dimensional statistical model of the Gaussian distribution with variance parameter is also presented in Section 6.

Since we only have to compute quotients of the likelihood or the gradient of the log-likelihood, we can omit the multiplicative constant which does not depend on θ.

As for the output of the neural networks, the activation function just before the output is different for each statistical model, the same can be said for the learning rate. In some cases, we apply an affine transformation on the variable θ to avoid divisions by zero during training. In every test case, we consider simple networks for an easier fine-tuning of the hyperparameters and also because the precise computation of the loss function is an important bottleneck.

For the initialization of the neural networks, biases are set to zero and weights are randomly sampled from a Gaussian distribution. As for the several hyperparameters, we take $N = 10$, $T = 50$ and $U = 1000$ unless stated otherwise. We take a latent space of dimension $p = 50$. For the posterior calculations, we keep the last $5 \cdot 10^4$ samples from the Markov chain over a total of 10^5 Metropolis-Hastings iterations. Increasing N is advised in order to get closer to the asymptotic case for the optimization problem, and increasing U and T is relevant for the precision of the Monte Carlo estimates. Nevertheless, this increases computation times and we have to do a trade-off between the former and the latter. As for the constrained optimization, we use $\nu = 2$, $M = 0.005$ and $\tilde{\eta}_{max} = 10^4$.

4.1 Multinomial model

The multinomial distribution can be interpreted as the generalization of the binomial distribution for higher dimensions. We denote : $X_i \sim \text{Multinomial}(n, (\theta_1, ..., \theta_q))$ with $n \in \mathbb{N}^*$, $\mathbf{X} \in \mathcal{X}^N$ and $\theta \in \Theta$, with : $\mathcal{X} = \{X \in \{0, ..., n\}^q \mid \sum_{j=1}^q X^j = n\}$ and $\Theta = \{\theta \in (0,1)^q \mid \sum_{j=1}^q \theta_j = 1\}$. We use $n = 10$ and $q = \dim(\theta) = 4$.

The likelihood function and the gradient of its logarithm are:

$$L_N(\mathbf{X} \mid \theta) = \prod_{i=1}^N \frac{n!}{X_i^1! \cdot ... \cdot X_i^q!} \prod_{j=1}^q \theta_j^{X_i^j} \propto \prod_{i=1}^N \prod_{j=1}^q \theta_j^{X_i^j}$$

$$\forall (i,j),\ \frac{\partial \log L}{\partial \theta_j}(X_i \mid \theta) = \frac{X_i^j}{\theta_j}.$$

The MLE is available : $\forall j,\ \hat{\theta}_{MLE}(j) = \frac{1}{nN} \sum_{i=1}^N X_i^j$ and the Jeffreys prior is the $\text{Dir}_q\left(\frac{1}{2}, ..., \frac{1}{2}\right)$ distribution, which is proper. The Jeffreys posterior is a conjugate Dirichlet distribution:

$$J_{post}(\theta \mid \mathbf{X}) = \text{Dir}_q(\theta; \gamma) \quad \text{with} \quad \gamma_j = \frac{1}{2} + \sum_{i=1}^N X_i^j.$$

We recall that the probability density function of a Dirichlet distribution is the following:

$$\mathrm{Dir}_q(x;\gamma) = \frac{\Gamma(\sum_{j=1}^q \gamma_j)}{\prod_{j=1}^q \Gamma(\gamma_j)} \prod_{j=1}^q x_j^{\gamma_j - 1}.$$

For approximating the RP, we opt for a simple neural network with one linear layer and a Softmax activation function assuring that all components are positive and sum to 1. Explicitly, we have that:

$$\theta = \mathrm{Softmax}(W^\top \varepsilon + b),$$

with $W \in \mathbb{R}^{p,4}$ the weight matrix and $b \in \mathbb{R}^4$ the bias vector. The density function of θ does not have a closed expression. The following results are obtained with $\alpha = 0.5$ for the divergence and the lower bound is used as the objective function.

```
# Parameters and classes
p = 50      # latent space dimension
q = 4       # parameter space dimension
n = 10      # multinomial parameter
N = 10      # number of data samples
J = 1000    # nb samples for MC estimation in MI
T = 50      # nb samples MC marginal likelihood
Multinom = torch_MultinomialModel(n=n, q=q)
input_size = p
output_size = q
n_samples_prior = 10**5
name_file = 'Multinomial_results.pkl'
file_path = os.path.join(path_plot, name_file)

low = 0.001          # lower bound (must verifiy q * low < 1)
upp = 1 - low*(q-1)  # upper bound (forced value with low and q)

seed_all(0)
NN = SingleLinear(input_size, output_size, m1=0, s1=0.1, b1=0, act1=nn.Softmax(dim=-1))
NN = AffineTransformation(NN, m_low=low, M_upp=upp)
VA = VA_NeuralNet(neural_net=NN, model=Multinom)
#Div = DivMetric_NeuralNet(va=VA, T=T, use_alpha=False, use_log_lik=True)
Div = DivMetric_NeuralNet(va=VA, T=T, use_alpha=True, alpha=0.5, use_log_lik=True, use_baseline=Fa

with torch.no_grad():
    theta_sample_init = Div.va.implicit_prior_sampler(n_samples_prior).numpy()

num_epochs = 10000
loss_fct = "LB_MI"
optimizer = torch_Adam
num_samples_MI = 200
freq_MI = 200
save_best_param = True
learning_rate = 0.0025

if not load_results_Multinom :
    ### Training loop
```

```
    MI, range_MI, lower_MI, upper_MI = Div.Partial_autograd(J, N, num_epochs, loss_fct, optimizer,
                                                            num_samples_MI, freq_MI, save_best_par
                                                            learning_rate, momentum=True)
    seed_all(0)
    with torch.no_grad():
        theta_sample = Div.va.implicit_prior_sampler(n_samples_prior).numpy()
        jeffreys_sample = Div.model.sample_Jeffreys(n_samples=n_samples_prior).numpy()
    all_params = torch.cat([param.view(-1) for param in NN.parameters()])

    seed_all(0)
    # Sample data from 'true' parameter
    theta_true = torch.tensor(q*[1/q])
    N = 10
    D = Multinom.sample(theta_true, N, 1)
    X = D[:, 0]

    # Posterior samples using Jeffreys prior
    n_samples_post = 5*10**4
    jeffreys_post = Multinom.sample_post_Jeffreys(X, n_samples_post)

    T_mcmc = 10**5 + 1
    sigma2_0 = torch.tensor(1.)
    eps_0 = torch.randn(p)
    eps_MH, batch_acc = VA.MH_posterior(eps_0, T_mcmc, sigma2_0, adap=True, Cov=True)
    theta_MH = NN(eps_MH)

    with torch.no_grad():
        theta_MH = theta_MH.numpy()
        jeffreys_post = jeffreys_post.numpy()
    theta_post = theta_MH[-n_samples_post:,:]

    # Saves all relevant quantities

    Mutual_Info = {'values' : MI, 'range' : range_MI, 'lower' : lower_MI, 'upper' : upper_MI}
    Prior = {'samples' : theta_sample, 'jeffreys' : jeffreys_sample, 'params' : all_params}
    Posterior = {'samples' : theta_post, 'jeffreys' : jeffreys_post}
    Multinom_results = {'MI' : Mutual_Info, 'prior' : Prior, 'post' : Posterior}
    with open(file_path, 'wb') as file:
        pickle.dump(Multinom_results, file)

else :
    with open(file_path, 'rb') as file:
        res = pickle.load(file)
        MI_dic = res['MI']
        Prior = res['prior']
        Posterior = res['post']
        MI, range_MI, lower_MI, upper_MI = MI_dic['values'], MI_dic['range'], MI_dic['lower'], MI_
        theta_sample, jeffreys_sample, all_params = Prior['samples'], Prior['jeffreys'], Prior['pa
        theta_post, jeffreys_post = Posterior['samples'], Posterior['jeffreys']
```

```
plt.figure(figsize=(5, 3))
plt.plot(range_MI, MI, '-', color=rougeCEA)
plt.plot(range_MI, MI, '*', color=rougeCEA)
# for i in range(len(range_MI)):
#     plt.plot([range_MI[i], range_MI[i]], [lower_MI[i], upper_MI[i]], color='lightgrey')
plt.plot(range_MI, upper_MI, '-', color='lightgrey')
plt.plot(range_MI, lower_MI, '-', color='lightgrey')
plt.fill_between(range_MI, lower_MI, upper_MI, color='lightgrey', alpha=0.5)
plt.xlabel(r"Epochs")
plt.ylabel("Generalized mutual information")
plt.grid()
#plt.yscale('log')
plt.tight_layout()
plt.show()
```

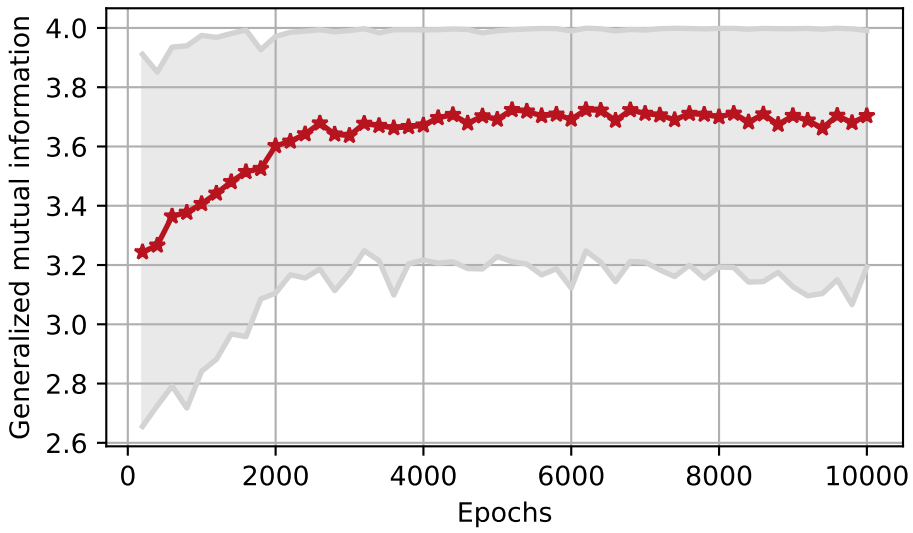

Figure 1: Monte Carlo estimation of the generalized mutual information with $\alpha = 0.5$ (from 200 samples) for π_{λ_e} where λ_e is the parameter of the neural network at epoch e. The red curve is the mean value and the gray zone is the 95% confidence interval. The learning rate used in the optimization is 0.0025.

```
fig, axs = plt.subplots(1, q, figsize=(15, 4))
for i in range(q):
    #axs[i].hist(theta_sample_init[:, i], density=True, bins="rice", color="red", label=r"Initial
    #axs[i].hist(theta_sample[:,i], density=True, bins="rice", label=r"Fitted prior", alpha=0.8, c
    axs[i].hist(theta_sample[:,i], density=True, bins="rice", label=r"Fitted prior", alpha=0.4)
    #axs[i].hist(jeffreys_sample[:,i], density=True, bins="rice", label="Jeffreys", alpha=0.6, col
    axs[i].hist(jeffreys_sample[:,i], density=True, bins="rice", label="Jeffreys", alpha=0.4, colo
    axs[i].set_xlabel(r"$\theta_{}$".format(i+1))
    axs[i].grid()
    axs[i].legend(fontsize=16)
plt.tight_layout()
plt.show()
```

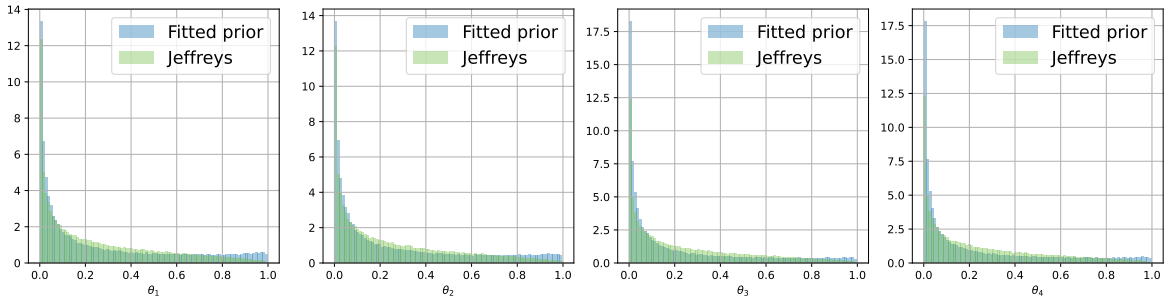

Figure 2: Histograms of the fitted prior and the Jeffreys prior Dir($\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}$) for each dimension of θ, each one is obtained from 10^5 samples.

For the posterior distribution, we sample 10 times from the Multinomial distribution using $\theta_{true} = (\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4})$. The covariance matrix in the proposition distribution of the Metropolis-Hastings algorithm is not diagonal, since we have a relation between the different components of θ, we introduce non-zero covariances. We also verified that the auto-correlation between the successive remaining samples of the Markov chain decreases rapidly on each component.

```
fig, axs = plt.subplots(1, q, figsize=(15, 4))
for i in range(q):
    axs[i].hist(jeffreys_post[:,i], density=True, bins="rice", label=r"Fitted post", alpha=0.4)
    axs[i].hist(theta_post[:,i], density=True, bins="rice", label=r"Jeffreys", alpha=0.4,color=ver
    axs[i].grid()
    axs[i].legend(fontsize=12)
    axs[i].set_xlabel(r"$\theta_{}$".format(i+1))
plt.tight_layout()
plt.show()
```

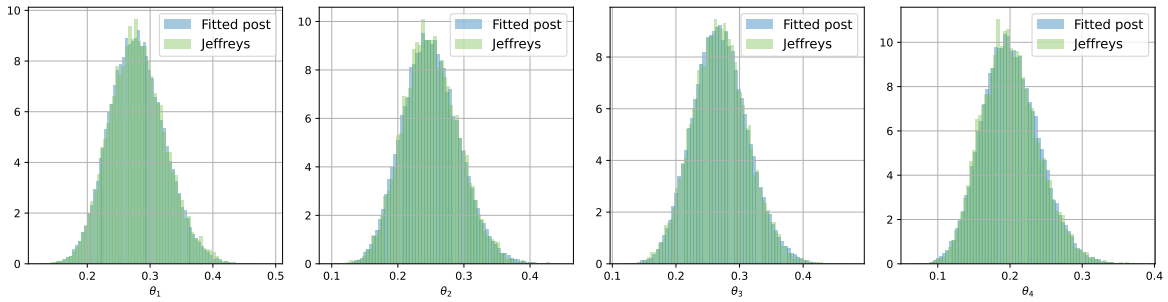

Figure 3: Histograms of the fitted posterior and the Jeffreys posterior for each dimension of θ, each one is obtained from $5 \cdot 10^4$ samples.

We notice (Figure 1) that the mutual information lies between 0 and $1/\alpha(1-\alpha) = 4$, which is coherent with the theory, the confidence interval is rather large, but the mean value has an increasing trend.

Although the shape of the fitted prior resembles the one of the Jeffreys prior, one can notice that it tends to put more weight towards the extremities of the interval (Figure 2). The posterior distribution however is quite similar to the target Jeffreys posterior on every component (Figure 3).

Since the multinomial model is simple and computationally practical, we would like to quantify the effect of the divergence with different α values on the output of the algorithm. In order to do so, we

utilize the maximum mean discrepancy (MMD) defined as :

$$\text{MMD}(p, q) = \|\mu_p - \mu_q\|_{\mathscr{H}},$$

where μ_p and μ_q are respectively the kernel mean embeddings of distributions p and q in a reproducible kernel Hilbert space (RKHS) $(\mathscr{H}, \|\cdot\|_{\mathscr{H}})$, meaning : $\mu_p(\theta') = \mathbb{E}_{\theta\sim p}[K(\theta, \theta')]$ for all $\theta' \in \Theta$ and K being the kernel. The MMD is used for instance in the context of two-sample tests Gretton et al. (2012), whose purpose is to compare distributions. We use in our computations the Gaussian or RBF kernel :

$$K(\theta, \theta') = \exp(-0.5 \cdot \|\theta - \theta'\|_2^2),$$

for which the MMD is a metric, this means that the following implication:

$$\text{MMD}(p, q) = 0 \implies p = q$$

is verified with the other axioms. In practice, we consider an unbiased estimator of the MMD^2 given by:

$$\widehat{\text{MMD}^2}(p, q) = \frac{1}{m(m-1)} \sum_{i\neq j} K(x_i, x_j) + \frac{1}{n(n-1)} \sum_{i\neq j} K(y_i, y_j) - \frac{2}{mn} \sum_{i,j} K(x_i, y_j),$$

where $(x_1, ..., x_m)$ and $(y_1, ..., y_n)$ are samples from p and q respectively. In our case, p is the distribution obtained through variational inference and q is the target Jeffreys distribution. Since the MMD can be time-consuming or memory inefficient to compute in practice for very large samples, we consider only the last $2 \cdot 10^4$ entries of our priors and posterior samples.

```
compute_autocorr = False
if compute_autocorr :
    def autocorrelation(x, lag):
        """Compute the autocorrelation of a 1D array at a given lag."""
        x = np.asarray(x)
        n = len(x)
        x_mean = np.mean(x)
        x_var = np.var(x)
        if x_var == 0:
            return 0
        return np.correlate(x - x_mean, x - x_mean, mode="full")[n - 1 + lag] / (n * x_var)
    lags = 100
    autocorrs = {i: [autocorrelation(theta_post[:, i], lag) for lag in range(lags + 1)] for i in r

    for i, ac in autocorrs.items():
        plt.plot(ac, label=f'Component {i+1}')
    plt.legend()
    plt.xlabel('Lag')
    plt.ylabel('Autocorrelation')
    plt.grid()
    plt.show()

name_file = 'Multinomial_MMD.pkl'
file_path = os.path.join(path_plot, name_file)
alphas = np.array([0.1, 0.25, 0.5, 0.75, 0.9])
len_alpha = len(alphas)
nb_samples_mmd = 2*10**4
```

```
if not load_results_MultiMMD :
    MMDvalues = np.zeros((len_alpha,2))
    for index_alpha in range(len_alpha):
        alpha = alphas[index_alpha]
        print(f'alpha value = {alpha}')
        # Parameters and classes
        p = 50      # latent space dimension
        q = 4       # parameter space dimension
        n = 10      # multinomial parameter
        N = 10      # number of data samples
        J = 1000    # nb samples for MC estimation in MI
        T = 50      # nb samples MC marginal likelihood
        Multinom = torch_MultinomialModel(n=n, q=q)
        input_size = p
        output_size = q
        n_samples_prior = 10**5
        low = 0.001           # lower bound (must verifiy q * low < 1)
        upp = 1 - low*(q-1)  # upper bound (forced value with low and q)

        ##### Prior #####
        seed_all(0)
        NN = SingleLinear(input_size, output_size, m1=0, s1=0.1, b1=0, act1=nn.Softmax(dim=-1))
        NN = AffineTransformation(NN, m_low=low, M_upp=upp)
        VA = VA_NeuralNet(neural_net=NN, model=Multinom)
        #Div = DivMetric_NeuralNet(va=VA, T=T, use_alpha=False, use_log_lik=True)
        Div = DivMetric_NeuralNet(va=VA, T=T, use_alpha=True, alpha=alpha, use_log_lik=True, use_b
        with torch.no_grad():
            theta_sample_init = Div.va.implicit_prior_sampler(n_samples_prior).numpy()

        num_epochs = 10000
        loss_fct = "LB_MI"
        optimizer = torch_Adam
        num_samples_MI = 200
        freq_MI = 200
        save_best_param = True
        learning_rate = 0.0025
        ### Training loop

        MI, range_MI, lower_MI, upper_MI = Div.Partial_autograd(J, N, num_epochs, loss_fct, optimi
                                                                num_samples_MI, freq_MI, save_best
                                                                momentum=True)


        theta_sample_torch = Div.va.implicit_prior_sampler(n_samples_prior)
        with torch.no_grad():
            theta_sample = theta_sample_torch.numpy()
        jeffreys_sample = Div.model.sample_Jeffreys(n_samples=n_samples_prior)

        #MMDvalues[index_alpha, 0] = np.sqrt(compute_MMD2(theta_sample, jeffreys_sample))
        MMDvalues[index_alpha, 0] = np.sqrt(MMD2_rbf(theta_sample[-nb_samples_mmd:,:], jeffreys_sa
```

```
        ##### Posterior ####
        seed_all(0)
        # Sample data from 'true' parameter
        theta_true = torch.tensor(q*[1/q])
        N = 10
        D = Multinom.sample(theta_true, N, 1)
        X = D[:, 0]
        # Posterior samples using Jeffreys prior
        n_samples_post = 5*10**4
        jeffreys_post = Multinom.sample_post_Jeffreys(X, n_samples_post)
        T_mcmc = 10**5 + 1
        sigma2_0 = torch.tensor(1.)
        eps_0 = torch.randn(p)
        eps_MH, batch_acc = VA.MH_posterior(eps_0, T_mcmc, sigma2_0, adap=True, Cov=True)
        theta_MH = NN(eps_MH)

        with torch.no_grad():
            theta_MH = theta_MH.numpy()
            jeffreys_post = jeffreys_post.numpy()
        theta_post = theta_MH[-n_samples_post:,:]

        #MMDvalues[index_alpha, 1] = np.sqrt(compute_MMD2(theta_post, jeffreys_post))
        MMDvalues[index_alpha, 1] = np.sqrt(MMD2_rbf(theta_post[-nb_samples_mmd:,:], jeffreys_post
    with open(file_path, 'wb') as file:
        pickle.dump(MMDvalues, file)
else :
    with open(file_path, 'rb') as file:
        MMDvalues = pickle.load(file)

data = np.column_stack((alphas, MMDvalues))
#headers = [" ", "Prior", "Posterior"]
headers = [r"$\alpha$", "Prior", "Posterior"]
def format_scientific(value, decimals=2):
    """Format a float in LaTeX-style scientific notation."""
    formatted = f"{value:.{decimals}e}"
    base, exp = formatted.split("e")
    return f"${base} \\times 10^{{{int(exp)}}}$"

formatted_data = [
    [f"${row[0]:.2f}$", format_scientific(row[1]), format_scientific(row[2])]
    for row in data
]
# formatted_data = [
#     [f"{row[0]:.2f}", f"{row[1]:.2e}", f"{row[2]:.3e}"] for row in data
# ]
table_data = [headers] + formatted_data
fig, ax = plt.subplots()
ax.axis("off")
table = ax.table(
```

```
    cellText=table_data,
    colLabels=None,
    loc="center",
    cellLoc="center",
)
for (i, key) in enumerate(headers):
    cell = table[0, i]
    cell.set_text_props(fontweight="bold")
    cell.set_facecolor("lightgray")
table.scale(1, 2.5)
table.auto_set_font_size(False)
table.set_fontsize(12)
plt.show()
```

α	**Prior**	**Posterior**
0.10	7.07×10^{-2}	2.09×10^{-3}
0.25	7.42×10^{-2}	3.39×10^{-3}
0.50	5.26×10^{-2}	1.96×10^{-3}
0.75	7.80×10^{-2}	1.50×10^{-3}
0.90	6.15×10^{-2}	4.84×10^{-4}

Figure 4: MMD values for different α-divergences at prior and posterior levels. As a reference on the prior level, when computing the criterion between two independent Dirichlet Dir$(\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2})$ distributions (ie the Jeffreys prior) on $2 \cdot 10^4$ samples, we obtain an order of magnitude of 10^{-3}.

According to Figure 4, the difference between α values in terms of the MMD criterion is essentially inconsequential. One remark is that the mutual information tends to be more unstable as α gets closer to 1. The explanation is that when α tends to 1, we have the approximation :

$$\hat{f}_\alpha(x) \approx \frac{x-1}{\alpha(\alpha-1)} + \frac{x\log(x)}{\alpha},$$

which diverges for all x because of the first term. Hence, we advise the user to avoid α values that are too close to 1. In the following, we use $\alpha = 0.5$ for the divergence.

```
compute_mmd_ref_value = False
if compute_mmd_ref_value :
    seed_all(0)
    dir1 = Multinom.sample_Jeffreys(n_samples=nb_samples_mmd)
    dir2 = Multinom.sample_Jeffreys(n_samples=nb_samples_mmd)
    print(f'MMD reference value : {np.sqrt(MMD2_rbf(dir1, dir2, gamma=0.5))}')
```

4.2 Probit model

We present in this section the probit model used to estimate seismic fragility curves, which was introduced by Kennedy et al. (1980), it is also referred as the log-normal model in the literature. A seismic fragility curve is the probability of failure $P_f(a)$ of a mechanical structure subjected to a seism as a function of a scalar value a derived from the seismic ground motion. The properties of the Jeffreys prior for this model are highlighted by Van Biesbroeck et al. (2024).

The model is defined by the observation of an i.i.d. sample $\mathbf{X} = (X_1, \dots, X_N)$ where for any i, $X_i \sim (Z, a) \in \mathcal{X} = \{0, 1\} \times (0, \infty)$. The distribution of the r.v. (Z, a) is parameterized by $\theta = (\theta_1, \theta_2) \in (0, \infty)^2$ as:

$$\begin{cases} a \sim \text{Log-}\mathcal{N}(\mu_a, \sigma_a^2) \\ P_f(a) = \Phi\left(\dfrac{\log a - \log \theta_1}{\theta_2}\right) \\ Z \sim \text{Bernoulli}(P_f(a)), \end{cases}$$

where Φ is the cumulative distribution function of the standard Gaussian. The probit function is the inverse of Φ. The likelihood is of the form :

$$\begin{cases} L_N(\mathbf{X} \,|\, \theta) = \displaystyle\prod_{i=1}^{N} p(a_i) \prod_{i=1}^{N} P_f(a_i)^{Z_i} (1 - P_f(a_i))^{1 - Z_i} \propto \prod_{i=1}^{N} P_f(a_i)^{Z_i} (1 - P_f(a_i))^{1 - Z_i} \\ p(a_i) = \dfrac{1}{a_i \sqrt{2\pi\sigma_a^2}} \exp\left(-\dfrac{1}{2\sigma_a^2} (\log a_i - \mu_a)^2\right). \end{cases}$$

For simplicity, we denote : $\gamma_i = \dfrac{\log a_i - \log \theta_1}{\theta_2} = \Phi^{-1}(P_f(a_i)) = \text{probit}(P_f(a_i))$, the gradient of the log-likelihood is the following :

$$\begin{cases} \dfrac{\partial \log L_N}{\partial \theta_1}(\mathbf{X} \,|\, \theta) = \displaystyle\sum_{i=1}^{N} \frac{1}{\theta_1 \theta_2} \left((-Z_i) \frac{\Phi'(\gamma_i)}{\Phi(\gamma_i)} + (1 - Z_i) \frac{\Phi'(\gamma_i)}{1 - \Phi(\gamma_i)} \right) \\ \dfrac{\partial \log L_N}{\partial \theta_2}(\mathbf{X} \,|\, \theta) = \displaystyle\sum_{i=1}^{N} \frac{\gamma_i}{\theta_2} \left((-Z_i) \frac{\Phi'(\gamma_i)}{\Phi(\gamma_i)} + (1 - Z_i) \frac{\Phi'(\gamma_i)}{1 - \Phi(\gamma_i)} \right). \end{cases}$$

There is no explicit formula for the MLE, so it has to be approximated using samples. This statistical model is a more difficult case than the previous one, since no explicit formula for the Jeffreys prior is available either but it has been shown by Van Biesbroeck et al. (2024) that it is improper in θ_2 and some asymptotic rates where derived. More precisely, when $\theta_1 > 0$ is fixed,

$$\begin{cases} J(\theta) \propto 1/\theta_2 & \text{as} \quad \theta_2 \longrightarrow 0 \\ J(\theta) \propto 1/\theta_2^3 & \text{as} \quad \theta_2 \longrightarrow +\infty. \end{cases}$$

If we fix $\theta_2 > 0$, the prior is proper for the variable θ_1 :

$$J(\theta) \propto \frac{|\log \theta_1|}{\theta_1} \exp\left(-\frac{(\log \theta_1 - \mu_a)^2}{2\theta_2 + 2\sigma_a^2}\right) \quad \text{when} \quad |\log \theta_1| \longrightarrow +\infty.$$

which resembles a log-normal distribution except for the $|\log \theta_1|$ factor. Since the density of the Jeffreys prior is not explicit and can not be computed directly, the Fisher information matrix is

computed in Van Biesbroeck et al. (2024) using numerical integration with Simpson's rule on a specific grid and then an interpolation is applied. We use this computation as the reference to evaluate the quality of the output of our algorithm. In the mentioned article, the posterior distribution is also computed with an adaptive Metropolis-Hastings algorithm on the variable θ, we refer to this algorithm as MH(θ) since it is different from the one mentioned in Section 3.4. More details on MH(θ) are given in Gauchy (2022). We take $\mu_a = 0$, $\sigma_a^2 = 1$, $N = 500$ and $U = 500$ for the computation of the prior.

As for the neural network, we use a one-layer network with an exp activation for θ_1 and a Softplus activation for θ_2. We have that :

$$\theta = \begin{pmatrix} \theta_1 \\ \theta_2 \end{pmatrix} = \begin{pmatrix} \exp(w_1^\top \varepsilon + b_1) \\ \log\left(1 + \exp(w_2^\top \varepsilon + b_2)\right) \end{pmatrix},$$

with $w_1, w_2 \in \mathbb{R}^p$ the weight vectors and $b_1, b_2 \in \mathbb{R}$ the biases, thus we have $\lambda = (w_1, w_2, b_1, b_2)$. Because this architecture remains simple, it is possible to elucidate the resulting marginal distributions of θ_1 and θ_2. The first component θ_1 follows a Log-$\mathcal{N}(b_1, \|w_1\|_2^2)$ distribution and θ_2 has an explicit density function :

$$p(\theta_2) = \frac{1}{\sqrt{2\pi\|w_2\|_2^2(1 - e^{-\theta_2})}} \exp\left(-\frac{1}{2\|w_2\|_2^2}\left(\log(e^{\theta_2} - 1) - b_2\right)^2\right).$$

These expressions describe the parameterized set $\mathscr{P}_\Lambda$ of priors considered in the optimization problem. This set is restrictive, so that the resulting VA-RP must be interpreted as the most objective —according to the mutual information criterion— prior among the ones in $\mathscr{P}_\Lambda$. Since we do not know any explicit expression of the Jeffreys prior for this prior, we cannot provide a precise comparison between the parameterized VA-RP elucidated above and the target. However, the form of the distribution of θ_1 qualitatively resembles its theoretical target. In the case of θ_2, the asymptotic decay rates of its density function can be derived:

$$\begin{cases} p(\theta_2) \underset{\theta_2 \to 0}{=} \frac{1}{\theta_2 \sqrt{2\pi}\|w_2\|_2} \exp\left(-\frac{(\log\theta_2 - b_2)^2}{2\|w_2\|_2^2}\right); \\ p(\theta_2) \underset{\theta_2 \to \infty}{=} \frac{1}{\sqrt{2\pi}\|w_2\|_2} \exp\left(-\frac{(\theta_2 - b_2)^2}{2\|w_2\|_2^2}\right). \end{cases} \tag{12}$$

While $\|w_2\|_2$ does not tend toward ∞, these decay rates strongly differ from the ones of the Jeffreys prior w.r.t. θ_2. Otherwise, the decay rates resemble to something proportional to $(\theta_2 + 1)^{-1}$ in both directions. In our numerical computations, the optimization process yielded a VA-RP with parameters w_2 and b_2 that did not diverge to extreme values.

```
# Parameters and classes
p = 50
q = 2
N = 500
J = 500
T = 50
input_size = p
output_size = q
low = 0.0001
upp = 1 + low
mu_a, sigma2_a = 0, 1
```

```
#mu_a, sigma2_a = 8.7 * 10**-3, 1.03
Probit = torch_ProbitModel(use_log_normal=True, mu_a=mu_a, sigma2_a=sigma2_a, set_beta=None, alt_s
n_samples_prior = 10**6
alpha = 0.5
name_file = 'Probit_results_unconstrained.pkl'
file_path = os.path.join(path_plot, name_file)

seed_all(0)
NN = SingleLinear(input_size, output_size, m1=0, s1=0.1, b1=0, act1=nn.Identity())
NN = DifferentActivations(NN, [torch.exp, nn.Softplus()])
NN = AffineTransformation(NN, low, upp)
VA = VA_NeuralNet(neural_net=NN, model=Probit)
#print(f'Number of parameters : {VA.nb_param}')
Div = DivMetric_NeuralNet(va=VA, T=T, use_alpha=True, alpha=alpha, use_log_lik=True)
with torch.no_grad():
    theta_sample_init = Div.va.implicit_prior_sampler(n_samples_prior).numpy()

num_epochs = 10000
loss_fct = "LB_MI"
optimizer = torch_Adam
num_samples_MI = 100
freq_MI = 500
save_best_param = True
learning_rate = 0.001

if not load_results_Probit_nocstr :
    MI, range_MI, lower_MI, upper_MI = Div.Partial_autograd(J, N, num_epochs, loss_fct, optimizer,
                                                            freq_MI, save_best_param, learning_rat
    all_params = torch.cat([param.view(-1) for param in NN.parameters()])

    seed_all(0)
    theta_sample = Div.va.implicit_prior_sampler(n_samples_prior)
    with torch.no_grad():
        theta_sample_prior = theta_sample.numpy()

    with open(tirages_path, 'rb') as file:
        data = pickle.load(file)
    data_A, data_Z = data[0], data[1]
    theta_true = np.array([3.37610525, 0.43304097])
    N = 50  # non-degenerate
    i = 2

    seed_all(0)
    Xstack = np.stack((data_Z[:N, i],data_A[:N, i]),axis=1)
    X = torch.tensor(Xstack)
    D = X.unsqueeze(1)
    Probit.data = D
    n_samples_post = 5000
    T_mcmc = 5*10**4 + 1
    sigma2_0 = torch.tensor(1.)
```

```
        eps_0 = torch.randn(p)
        #eps_0 = 10 * torch.ones(p)
        eps_MH, batch_acc = VA.MH_posterior(eps_0, T_mcmc, sigma2_0, target_accept=0.4, adap=True, Cov
        theta_MH = NN(eps_MH)
        with torch.no_grad():
            theta_MH = theta_MH.detach().numpy()
        theta_post_nocstr = theta_MH[-n_samples_post:,-n_samples_post:]

        # Saves all relevant quantities
        Mutual_Info = {'values' : MI, 'range' : range_MI, 'lower' : lower_MI, 'upper' : upper_MI}
        Prior = {'samples' : theta_sample_prior, 'jeffreys' : None, 'params' : all_params}
        Posterior = {'samples' : theta_post_nocstr, 'jeffreys' : None}
        Probit_results_nocstr = {'MI' : Mutual_Info, 'prior' : Prior, 'post' : Posterior}
        with open(file_path, 'wb') as file:
            pickle.dump(Probit_results_nocstr, file)

else :
    with open(file_path, 'rb') as file:
        res = pickle.load(file)
        MI_dic = res['MI']
        Prior = res['prior']
        Posterior = res['post']
        MI, range_MI, lower_MI, upper_MI = MI_dic['values'], MI_dic['range'], MI_dic['lower'], MI_
        theta_sample_prior, jeffreys_sample, all_params = Prior['samples'], Prior['jeffreys'], Pri
        theta_post_nocstr, jeffreys_post = Posterior['samples'], Posterior['jeffreys']

plt.figure(figsize=(4, 3))
plt.plot(range_MI, MI, '-', color=rougeCEA)
plt.plot(range_MI, MI, '*', color=rougeCEA)
# for i in range(len(range_MI)):
#     plt.plot([range_MI[i], range_MI[i]], [lower_MI[i], upper_MI[i]], color='black')
plt.plot(range_MI, upper_MI, '-', color='lightgrey')
plt.plot(range_MI, lower_MI, '-', color='lightgrey')
plt.fill_between(range_MI, lower_MI, upper_MI, color='lightgrey', alpha=0.5)
plt.xlabel(r"Epochs")
plt.ylabel(r"Generalized mutual information")
plt.grid()
#plt.yscale('log')
plt.tight_layout()
plt.show()
```

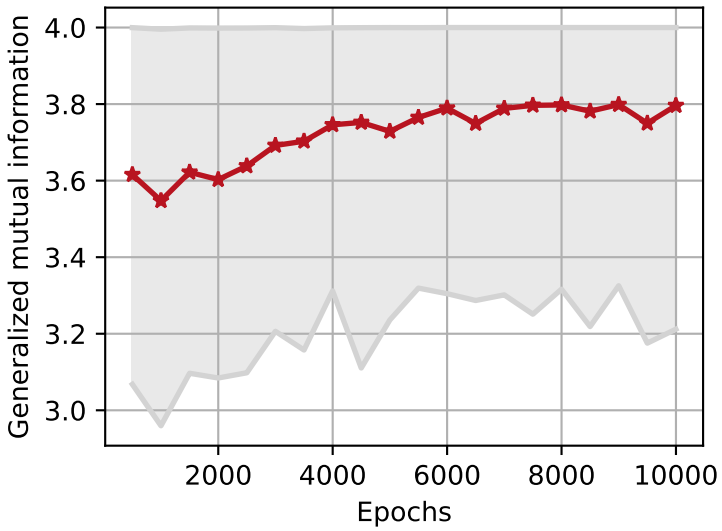

Figure 5: Monte Carlo estimation of the generalized mutual information with $\alpha = 0.5$ (from 100 samples) for π_{λ_e} where λ_e is the parameter of the neural network at epoch e. The red curve is the mean value and the gray zone is the 95% confidence interval. The learning rate used in the optimization is 0.001.

In Figure 5 is shown the evolution of the mutual information through the optimization of the VA-RP for the probit model. We perceive high mutual information values at the initialization, which we interpret as a result of the fact that the parametric prior on θ_1 is already quite close to its target distribution.

With α-divergences, using a moment constraint of the form $a(\theta_2) = \theta_2^\kappa$ for the second component is relevant here as long as $\kappa \in \left(0, \frac{2}{1+1/\alpha}\right)$, to ensure that the resulting constrained prior is indeed proper. With $\alpha = 0.5$, we take the value $\kappa = 1/8$ and we use the same neural network. The evolution of the mutual information through the optimization of the constrained VA-RP is proposed in Figure 6. In Figure 7 is presented the evolution of the constrained gap: the difference between the target and current values for the constraint.

```
kappa = 1/8
K_val = np.mean((theta_sample_prior[:,1]**kappa)**(1/alpha))
c_val = np.mean((theta_sample_prior[:,1]**kappa)**(1+1/alpha))
constr_val = c_val / K_val
alpha_constr = np.mean((theta_sample_prior[:,0]**-kappa + 1)**-1)
#print(f'Constraint value estimation : {constr_val}')  # = 0.839594841003418

# Parameters and classes
p = 50      # latent space dimension
q = 2       # parameter space dimension
N = 500     # number of data samples
J = 500     # nb samples for MC estimation in MI
T = 50      # nb samples MC marginal likelihood
alpha = 0.5
input_size = p
output_size = q
low = 0.0001          # lower bound
```

```
upp = 1 + low
n_samples_prior = 10**6

mu_a, sigma2_a = 0, 1
#mu_a, sigma2_a = 8.7 * 10**-3, 1.03
Probit = torch_ProbitModel(use_log_normal=True, mu_a=mu_a, sigma2_a=sigma2_a, set_beta=None, alt_s

# Constraints
beta = torch.tensor([kappa])
b = torch.tensor([[alpha_constr,constr_val]])
T_cstr = 100000
eta_augm = torch.tensor([[0.,1.]])
eta = torch.tensor([[0.,1.]])
name_file = 'Probit_results_constrained.pkl'
file_path = os.path.join(path_plot, name_file)

seed_all(0)
NN = SingleLinear(input_size, output_size, m1=0, s1=0.1, b1=0, act1=nn.Identity())
NN = DifferentActivations(NN, [torch.exp, nn.Softplus()])
NN = AffineTransformation(NN, low, upp)
VA = VA_NeuralNet(neural_net=NN, model=Probit)
#print(f'Number of parameters : {VA.nb_param}')
Div = DivMetric_NeuralNet(va=VA, T=T, use_alpha=True, alpha=0.5, use_log_lik=True)
Constr = Constraints_NeuralNet(div=Div, betas=beta, b=b, T_cstr=T_cstr,
                               objective='LB_MI', lag_method='augmented', eta_augm=eta_augm, rule=

with torch.no_grad():
    theta_sample_init = Div.va.implicit_prior_sampler(n_samples_prior).numpy()

num_epochs = 10000
optimizer = torch_Adam
num_samples_MI = 100
freq_MI = 500
save_best_param = False
learning_rate = 0.0005
freq_augm = 100

if not load_results_Probit_cstr :

    ### Training loop
    MI, constr_values, range_MI, lower_MI, upper_MI = Constr.Partial_autograd(J, N, eta, num_epoch
                                                                              freq_MI, save_best_param, learning_rat
                                                                              freq_augm=freq_augm, max_violation=0.0

    constr_values = torch.stack(constr_values).numpy()
    all_params = torch.cat([param.view(-1) for param in NN.parameters()])
    seed_all(0)
    with torch.no_grad():
        theta_sample = Constr.va.implicit_prior_sampler(n_samples_prior).numpy()
```

```
    print(f'Moment of order {beta.item()}, estimation : {np.mean(theta_sample**beta.item(),axis=0)}

    seed_all(0)
    with open(tirages_path, 'rb') as file:
        data = pickle.load(file)
    data_A, data_Z = data[0], data[1]
    N = 50  # non-degenerate
    i = 2

    Xstack = np.stack((data_Z[:N, i],data_A[:N, i]),axis=1)
    X = torch.tensor(Xstack)
    D = X.unsqueeze(1)
    Probit.data = D
    n_samples_post = 5000
    T_mcmc = 5*10**4 + 1
    sigma2_0 = torch.tensor(1.)
    eps_0 = torch.randn(p)
    eps_MH, batch_acc = VA.MH_posterior(eps_0, T_mcmc, sigma2_0, target_accept=0.4, adap=True, Cov
    theta_MH = NN(eps_MH)
    with torch.no_grad():
        theta_MH = theta_MH.detach().numpy()
    theta_post_cstr = theta_MH[-n_samples_post:,-n_samples_post:]

    # Saves all relevant quantities
    Mutual_Info = {'values' : MI, 'range' : range_MI, 'lower' : lower_MI, 'upper' : upper_MI, 'cons
    Prior = {'samples' : theta_sample_prior, 'jeffreys' : None, 'params' : all_params}
    Posterior = {'samples' : theta_post_cstr, 'jeffreys' : None}
    Probit_results_cstr = {'MI' : Mutual_Info, 'prior' : Prior, 'post' : Posterior}
    with open(file_path, 'wb') as file:
        pickle.dump(Probit_results_cstr, file)

else :
    with open(file_path, 'rb') as file:
        res = pickle.load(file)
        MI_dic = res['MI']
        Prior = res['prior']
        Posterior = res['post']
        MI, range_MI, lower_MI, upper_MI, constr_values = MI_dic['values'], MI_dic['range'], MI_di
        theta_sample_prior, jeffreys_sample, all_params = Prior['samples'], Prior['jeffreys'], Pri
        theta_post_cstr, jeffreys_post = Posterior['samples'], Posterior['jeffreys']

plt.figure(figsize=(5, 3))
plt.plot(range_MI, MI, '-', color=rougeCEA)
plt.plot(range_MI, MI, '*', color=rougeCEA)
# for i in range(len(range_MI)):
#     plt.plot([range_MI[i], range_MI[i]], [lower_MI[i], upper_MI[i]], color='black')
plt.plot(range_MI, upper_MI, '-', color='lightgrey')
plt.plot(range_MI, lower_MI, '-', color='lightgrey')
plt.fill_between(range_MI, lower_MI, upper_MI, color='lightgrey', alpha=0.5)
plt.xlabel(r"Epochs")
```

```
plt.ylabel(r"Generalized mutual information")
plt.grid()
#plt.yscale('log')
plt.tight_layout()
plt.show()
```

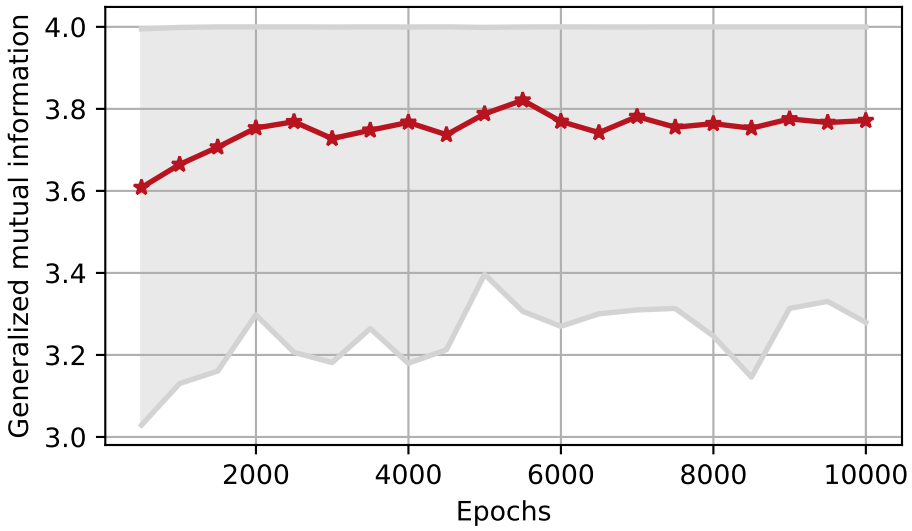

Figure 6: Monte Carlo estimation of the generalized mutual information with $\alpha = 0.5$ (from 100 samples) for π_{λ_e} where λ_e is the parameter of the neural network at epoch e. The red curve is the mean value and the gray zone is the 95% confidence interval. The learning rate used in the optimization is 0.0005.

```
plt.figure(figsize=(5, 3))
plt.plot(range_MI, constr_values[:,0,1], '-', color=vertCEA)
plt.plot(range_MI, constr_values[:,0,1], '*', color=vertCEA)
plt.xlabel(r"Epochs")
plt.ylabel(r"Constraint gap")
plt.grid()
#plt.yscale('log')
plt.tight_layout()
plt.show()
```

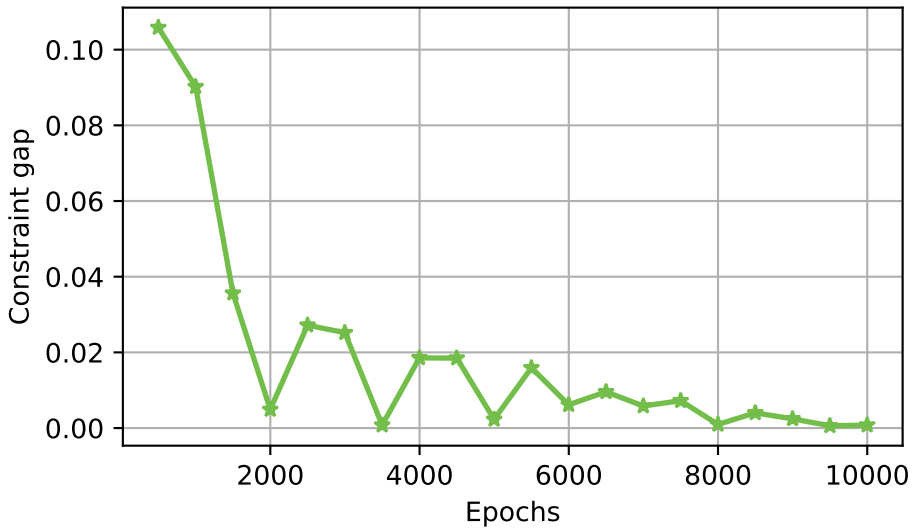

Figure 7: Evolution of the constraint value gap during training. It corresponds to the difference between the target and current values for the constraint (in absolute value)

For the posterior, we take as dataset 50 samples from the probit model. For computational reasons, the Metropolis-Hastings algorithm is applied for only $5 \cdot 10^4$ iterations. An important remark is that if the size of the dataset is rather small, the probability that the data is degenerate is not negligible. By degenerate data, we refer to situations when the data points are partitioned into two disjoint subsets when classified according to a values, the posterior becomes improper because the likelihood is constant (Van Biesbroeck et al. (2024)). In such cases, the convergence of the Markov chains is less apparent, the plots for this section are obtained with non-degenerate datasets.

```
N = 50  # non-degenerate
i = 2
file_path = os.path.join(int_jeffreys_path, f'model_J_{i}')
with open(file_path, 'rb') as file:
    model_J = pickle.load(file)
theta_J = model_J['logs']['post'][N]

x1 = theta_post_nocstr[:, 0]
y1 = theta_post_nocstr[:, 1]
x2 = theta_J[:, 0]
y2 = theta_J[:, 1]

kde_x1 = gaussian_kde(x1)
kde_y1 = gaussian_kde(y1)
kde_x2 = gaussian_kde(x2)
kde_y2 = gaussian_kde(y2)

# Create figure and gridspec layout
fig = plt.figure(figsize=(8, 6))
gs = gridspec.GridSpec(4, 5)

# Main scatter plot
ax_main = fig.add_subplot(gs[1:4, 0:3])
```

```
ax_main.scatter(x1, y1, alpha=0.5, s=20, marker='.', label='Fitted posterior',zorder=2)
ax_main.scatter(x2, y2, alpha=0.5, s=20, marker='.', label='Jeffreys posterior',zorder=1, color=ver
ax_main.set_xlabel(r'$\theta_1$')
ax_main.set_ylabel(r'$\theta_2$')
ax_main.legend(loc='upper left', fontsize=11)
ax_main.grid()

# Marginal histogram / KDE for alpha
ax_x_hist = fig.add_subplot(gs[0, 0:3], sharex=ax_main)
ax_x_hist.hist(x1, bins='rice', alpha=0.4, label='Fitted histogram',density=True)
ax_x_hist.hist(x2, bins='rice', alpha=0.4, label='Jeffreys histogram',density=True,color=vertCEA)
x_vals = np.linspace(min(x1.min(), x2.min()), max(x1.max(), x2.max()), 100)
ax_x_hist.plot(x_vals, kde_x1(x_vals), color='blue', lw=2, label='Fitted KDE')
ax_x_hist.plot(x_vals, kde_x2(x_vals), color='green', lw=2, label='Jeffreys KDE')
ax_x_hist.set_ylabel(r'Marginal $\theta_1$')
ax_x_hist.tick_params(axis='x', labelbottom=False)
ax_x_hist.legend()
ax_x_hist.grid()

# Marginal histogram / KDE for beta
ax_y_hist = fig.add_subplot(gs[1:4, 3:5], sharey=ax_main)
ax_y_hist.hist(y1, bins='rice', orientation='horizontal', alpha=0.4, label='Fitted histogram',dens
ax_y_hist.hist(y2, bins='rice', orientation='horizontal', alpha=0.4, label='Jeffreys histogram',de
y_vals = np.linspace(min(y1.min(), y2.min()), max(y1.max(), y2.max()), 100)
ax_y_hist.plot(kde_y1(y_vals), y_vals, color='blue', lw=2, label='Fitted KDE')
ax_y_hist.plot(kde_y2(y_vals), y_vals, color='green', lw=2, label='Jeffreys KDE')
ax_y_hist.set_xlabel(r'Marginal $\theta_2$')
ax_y_hist.tick_params(axis='y', labelleft=False)
ax_y_hist.legend()
ax_y_hist.grid()

plt.tight_layout()
plt.show()
```

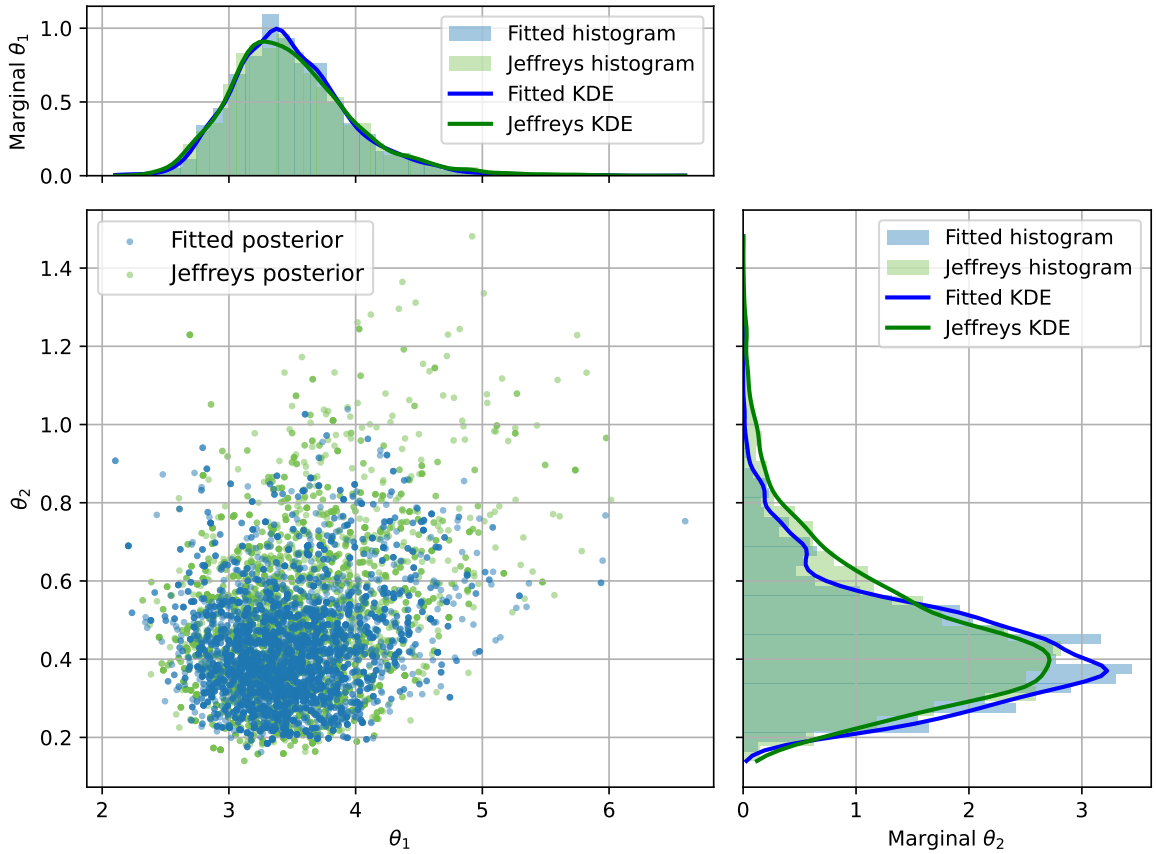

Figure 8: Scatter histogram of the unconstrained fitted posterior and the Jeffreys posterior distributions obtained from 5000 samples. Kernel density estimation is used on the marginal distributions in order to approximate their density functions with Gaussian kernels.

As Figure 8 shows, we obtain a relevant approximation of the true Jeffreys posterior especially on the variable θ_1, whereas a small difference is present for the tail of the distribution on θ_2. The latter remark was expected regarding the analytical study of the marginal distribution of π_λ w.r.t. θ_2 given the architecture considered for the VA-RP (see Equation 12). It is interesting to see that the difference between the posteriors is harder to discern in the neighborhood of $\theta_2 = 0$. Indeed, in such case where the data are not degenerate, the likelihood provides a strong decay rate when $\theta_2 \to 0$ that makes the influence of the prior negligible (see Van Biesbroeck et al. (2024)):

$$L_N(\mathbf{X}\,|\,\theta) \underset{\theta_2 \to 0}{=} \theta_2^{\|\chi\|_2^2} \exp\left(-\frac{1}{2\theta_2^2}\sum_{i=1}^{N}\chi_i(\log a_i - \log\theta_1)^2\right),$$

where $\chi \in \{0,1\}^N$ is a non-null vector that depends on $\mathbf{X}$.

When $\theta_2 \to \infty$, however, the likelihood does not reduce the influence of the prior has it remains asymptotically constant: $L_N(\mathbf{X}\,|\,\theta) \underset{\theta_2 \to \infty}{\to} 2^{-N}$.

```
N = 50  # non-degenerate
i = 2
file_path = os.path.join(int_jeffreys_path, f'model_J_constraint_{i}')
with open(file_path, 'rb') as file:
    model_J = pickle.load(file)
theta_J = model_J['logs']['post'][N]
```

```
x1 = theta_post_cstr[:, 0]
y1 = theta_post_cstr[:, 1]
x2 = theta_J[:, 0]
y2 = theta_J[:, 1]
kde_x1 = gaussian_kde(x1)
kde_y1 = gaussian_kde(y1)
kde_x2 = gaussian_kde(x2)
kde_y2 = gaussian_kde(y2)

# Create figure and gridspec layout
fig = plt.figure(figsize=(8, 6))
gs = gridspec.GridSpec(4, 5)

# Main scatter plot
ax_main = fig.add_subplot(gs[1:4, 0:3])
ax_main.scatter(x1, y1, alpha=0.5, s=20, marker='.', label='Fitted posterior',zorder=2)
ax_main.scatter(x2, y2, alpha=0.5, s=20, marker='.', label='Jeffreys posterior',zorder=1, color=ver
ax_main.set_xlabel(r'$\theta_1$')
ax_main.set_ylabel(r'$\theta_2$')
ax_main.legend(loc='upper left',fontsize=11)
ax_main.grid()

# Marginal histogram / KDE for alpha
ax_x_hist = fig.add_subplot(gs[0, 0:3], sharex=ax_main)
ax_x_hist.hist(x1, bins='rice', alpha=0.4, label='Fitted histogram',density=True)
ax_x_hist.hist(x2, bins='rice', alpha=0.4, label='Jeffreys histogram',density=True,color=vertCEA)
x_vals = np.linspace(min(x1.min(), x2.min()), max(x1.max(), x2.max()), 100)
ax_x_hist.plot(x_vals, kde_x1(x_vals), color='blue', lw=2, label='Fitted KDE')
ax_x_hist.plot(x_vals, kde_x2(x_vals), color='green', lw=2, label='Jeffreys KDE')
ax_x_hist.set_ylabel(r'Marginal $\theta_1$')
ax_x_hist.tick_params(axis='x', labelbottom=False)
ax_x_hist.legend()
ax_x_hist.grid()

# Marginal histogram / KDE for beta
ax_y_hist = fig.add_subplot(gs[1:4, 3:5], sharey=ax_main)
ax_y_hist.hist(y1, bins='rice', orientation='horizontal', alpha=0.4, label='Fitted histogram',dens
ax_y_hist.hist(y2, bins='rice', orientation='horizontal', alpha=0.4, label='Jeffreys histogram',de
y_vals = np.linspace(min(y1.min(), y2.min()), max(y1.max(), y2.max()), 100)
ax_y_hist.plot(kde_y1(y_vals), y_vals, color='blue', lw=2, label='Fitted KDE')
ax_y_hist.plot(kde_y2(y_vals), y_vals, color='green', lw=2, label='Jeffreys KDE')
ax_y_hist.set_xlabel(r'Marginal $\theta_2$')
ax_y_hist.tick_params(axis='y', labelleft=False)
ax_y_hist.legend()
ax_y_hist.grid()

plt.tight_layout()
plt.show()
```

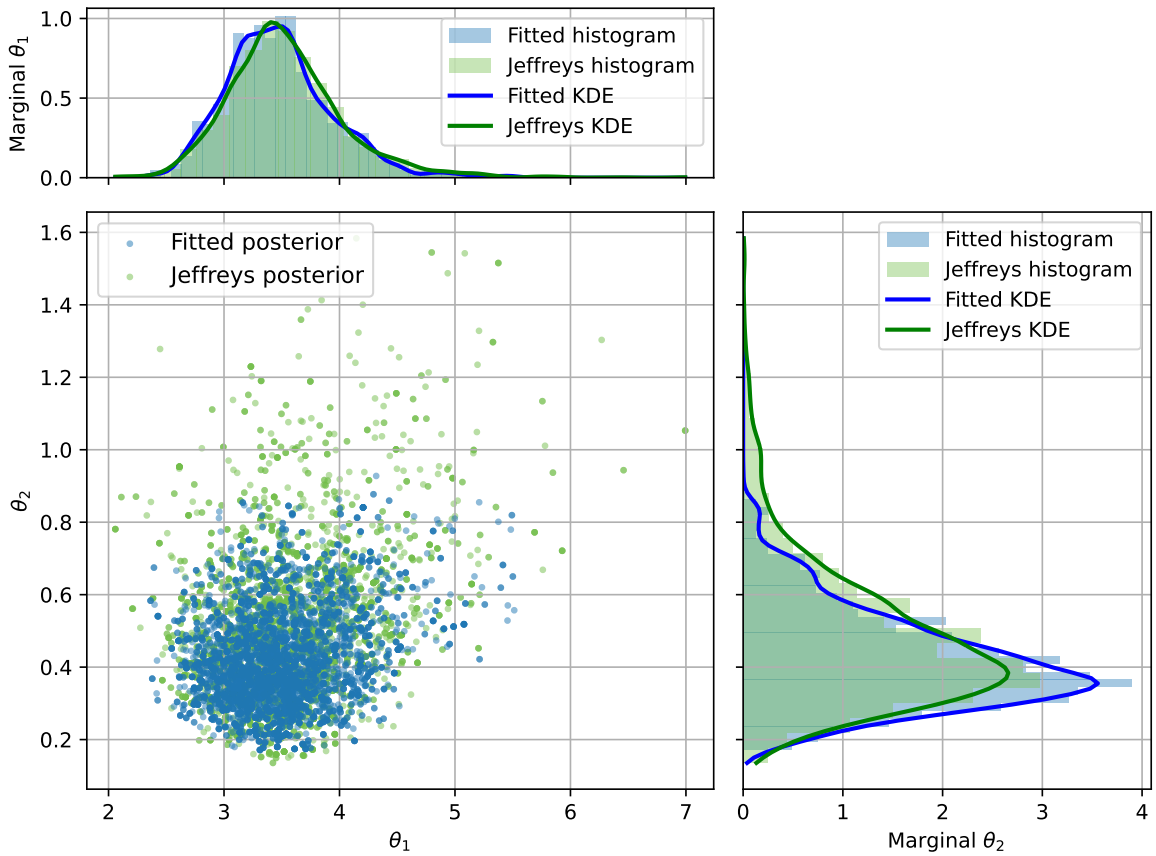

Figure 9: Scatter histogram of the constrained fitted posterior and the target posterior distributions obtained from 5000 samples. Kernel density estimation is used on the marginal distributions in order to approximate their density functions with Gaussian kernels.

The result on the constrained case (Figure 9) is very similar to the unconstrained one.

Indeed, the priors had already comparable shapes. Altogether, one can observe that the variational inference approach yields close results to the numerical integration approach Van Biesbroeck et al. (2024), with or without constraints, even tough the matching of the decay rates w.r.t. θ_2 remains limited given the simple network that we have used in this case.

To ascertain the relevancy of our posterior approximation, we compute the posterior mean euclidean norm difference $\mathbb{E}_\theta[||\theta - \theta_{true}||]$ as a function of the size of the dataset. In each computation, the neural network remains the same but the dataset changes by adding new entries.

Furthermore, in order to assess the stability of the stochastic optimization with respect to the random number generator (RNG) seed, we also compute the empirical cumulative distribution functions (ECDFs) for each posterior distribution. For every seed, the parameters of the neural network are expected to be different, we keep the same dataset for the MCMC sampling however.

Both types of computations are done in the unconstrained case as well as the constrained one. The different plots and details can be found in Section 6.

5 Conclusion

In this work, we developed an algorithm to perform variational approximation of reference priors using a generalized definition of mutual information based on f-divergences. To enhance computational efficiency, we derived a lower bound of the generalized mutual information. Additionally, because reference priors often yield improper posteriors, we adapted the variational definition of the problem to incorporate constraints that ensure the posteriors are proper.

Numerical experiments have been carried out on various test cases of different complexities in order to validate our approach. These test cases range from purely toy models to more real-world problems, namely the estimation of seismic fragility curve parameters using a probit statistical model.

The results demonstrate the usefulness of our approach in estimating both prior and posterior distributions across various problems.

Our development is supported by an open source and flexible implementation, which can be adapted to a wide range of statistical models.

Looking forward, the approximation of the tails of the reference priors could be improved. That is a complex problem in the field of variational approximation, as well as the stability of the algorithm when using deeper networks. An extension of this work to the approximation of Maximal Data Information (MDI) priors is also appealing, thanks to the fact MDI are proper under certain assumptions precised in Bousquet (2008).

Acknowledgement

This research was supported by the CEA (French Alternative Energies and Atomic Energy Commission) and the SEISM Institute (https://www.institut-seism.fr/en/).

6 Appendix

6.1 Gradient computation of the generalized mutual information

We recall that $F(x) = f(x) - xf'(x)$ and p_λ is a shortcut notation for $p_{\pi_\lambda,N}$ being the marginal distribution under π_λ. The generalized mutual information writes :

$$\begin{aligned} I_{\mathrm{D}_f}(\pi_\lambda; L_N) &= \int_\Theta \mathrm{D}_f(p_\lambda \| L_N(\cdot\,|\,\theta))\pi_\lambda(\theta)d\theta \\ &= \int_\Theta \int_{\mathcal{X}^N} \pi_\lambda(\theta) L_N(\mathbf{X}\,|\,\theta) f\left(\frac{p_\lambda(\mathbf{X})}{L_N(\mathbf{X}\,|\,\theta)}\right) d\mathbf{X}d\theta. \end{aligned}$$

For each l, taking the derivative with respect to λ_l yields :

$$\begin{aligned} \frac{\partial I_{\mathrm{D}_f}}{\partial \lambda_l}(\pi_\lambda; L_N) = &\int_\Theta \int_{\mathcal{X}^N} \frac{\partial \pi_\lambda}{\partial \lambda_l}(\theta) L_N(\mathbf{X}\,|\,\theta) f\left(\frac{p_\lambda(\mathbf{X})}{L_N(\mathbf{X}\,|\,\theta)}\right) d\mathbf{X}d\theta \\ &+ \int_\Theta \int_{\mathcal{X}^N} \pi_\lambda(\theta) L_N(\mathbf{X}\,|\,\theta) \frac{\partial p_\lambda}{\partial \lambda_l} \frac{1}{L_N(\mathbf{X}\,|\,\theta)}(\mathbf{X}) f'\left(\frac{p_\lambda(\mathbf{X})}{L_N(\mathbf{X}\,|\,\theta)}\right) d\mathbf{X}d\theta, \end{aligned}$$

or in terms of expectations :

$$\frac{\partial I_{\mathrm{D}_f}}{\partial \lambda_l}(\pi_\lambda; L_N) = \frac{\partial}{\partial \lambda_l} \mathbb{E}_{\theta\sim\pi_\lambda}\left[\tilde{I}(\theta)\right] + \mathbb{E}_{\theta\sim\pi_\lambda}\left[\mathbb{E}_{\mathbf{X}\sim L_N(\cdot|\theta)}\left[\frac{1}{L_N(\mathbf{X}\,|\,\theta)} \frac{\partial p_\lambda}{\partial \lambda_l}(\mathbf{X}) f'\left(\frac{p_\lambda(\mathbf{X})}{L_N(\mathbf{X}\,|\,\theta)}\right)\right]\right],$$

where :

$$\tilde{I}(\theta) = \int_{\mathcal{X}^N} L_N(\mathbf{X}\,|\,\theta) f\left(\frac{p_\lambda(\mathbf{X})}{L_N(\mathbf{X}\,|\,\theta)}\right) d\mathbf{X}.$$

We note that the derivative with respect to λ_l does not apply to $\tilde{I}$ in the previous equation. Using the chain rule yields :

$$\frac{\partial}{\partial \lambda_l}\mathbb{E}_{\theta\sim\pi_\lambda}\left[\tilde{I}(\theta)\right] = \frac{\partial}{\partial \lambda_l}\mathbb{E}_\varepsilon\left[\tilde{I}(g(\lambda,\varepsilon))\right] = \mathbb{E}_\varepsilon\left[\sum_{j=1}^{q}\frac{\partial \tilde{I}}{\partial \theta_j}(g(\lambda,\varepsilon))\frac{\partial g_j}{\partial \lambda_l}(\lambda,\varepsilon)\right].$$

We have the following for every $j \in \{1, ..., q\}$:

$$\begin{aligned}
\frac{\partial \tilde{I}}{\partial \theta_j}(\theta) &= \int_{\mathcal{X}^N} \frac{-p_\lambda(\mathbf{X})}{L_N(\mathbf{X}\,|\,\theta)}\frac{\partial L_N}{\partial \theta_j}(\mathbf{X}\,|\,\theta) f'\left(\frac{p_\lambda(\mathbf{X})}{L_N(\mathbf{X}\,|\,\theta)}\right) + f\left(\frac{p_\lambda(\mathbf{X})}{L_N(\mathbf{X}\,|\,\theta)}\right)\frac{\partial L_N}{\partial \theta_j}(\mathbf{X}\,|\,\theta) d\mathbf{X} \\
&= \int_{\mathcal{X}^N} F\left(\frac{p_\lambda(\mathbf{X})}{L_N(\mathbf{X}\,|\,\theta)}\right)\frac{\partial L_N}{\partial \theta_j}(\mathbf{X}\,|\,\theta) d\mathbf{X} \\
&= \mathbb{E}_{\mathbf{X}\sim L_N(\cdot|\theta)}\left[\frac{\partial \log L_N}{\partial \theta_j}(\mathbf{X}\,|\,\theta) F\left(\frac{p_\lambda(\mathbf{X})}{L_N(\mathbf{X}\,|\,\theta)}\right)\right].
\end{aligned}$$

Putting everything together, we finally obtain the desired formula. The gradient of the generalized lower bound function is obtained in a very similar manner.

6.2 Gaussian distribution with variance parameter

We consider a normal distribution where θ is the variance parameter : $X_i \sim \mathcal{N}(\mu, \theta)$ with $\mu \in \mathbb{R}$, $\mathbf{X} \in \mathcal{X}^N = \mathbb{R}^N$ and $\theta \in \mathbb{R}_+^*$. We take $\mu = 0$. The likelihood and score functions are :

$$L_N(\mathbf{X}\,|\,\theta) = \prod_{i=1}^{N}\frac{1}{\sqrt{2\pi\theta}}\exp\left(-\frac{1}{2\theta}(X_i-\mu)^2\right)$$

$$\frac{\partial \log L_N}{\partial \theta}(\mathbf{X}\,|\,\theta) = -\frac{N}{2\theta} + \frac{1}{2\theta^2}\sum_{i=1}^{N}(X_i-\mu)^2.$$

The MLE is available : $\hat{\theta}_{MLE} = \frac{1}{N}\sum_{i=1}^{N} X_i$. However, the Jeffreys prior is an improper distribution in this case : $J(\theta) \propto 1/\theta$. Nevertheless, the Jeffreys posterior is a proper inverse-gamma distribution:

$$J_{post}(\theta|\,\mathbf{X}) = \Gamma^{-1}\left(\theta; \frac{N}{2}, \frac{1}{2}\sum_{i=1}^{N}(X_i-\mu)^2\right).$$

We use a neural network with one layer and a Softplus activation function. The dimension of the latent variable ε is $p = 10$.

```
# Parameters and classes
p = 10     # latent space dimension
q = 1      # theta dimension
mu = 0     # normal parameter (mean)
N = 10     # number of data samples
J = 1000   # nb samples for MC estimation in MI
T = 50     # nb samples MC marginal likelihood
```

```
Normal = torch_NormalModel_variance(mu=mu)
low = 0.0001        # lower bound
upp = 1 + low       # upper bound (not relevant here)
input_size = p
output_size = 1
n_samples_prior = 10**5
name_file = 'Normal_results_unconstrained.pkl'
file_path = os.path.join(path_plot, name_file)

seed_all(0)
NN = SingleLinear(input_size, output_size, m1=0, s1=0.1, b1=0, act1=nn.Softplus())
NN = AffineTransformation(NN, m_low=low, M_upp=upp)
VA = VA_NeuralNet(neural_net=NN, model=Normal)
#print(f'Number of parameters : {VA.nb_param}')
#Div = DivMetric_NeuralNet(va=VA, T=T, use_alpha=False, use_log_lik=True)
Div = DivMetric_NeuralNet(va=VA, T=T, use_alpha=True, alpha=0.5, use_log_lik=True, use_baseline=Fa

with torch.no_grad():
    theta_sample_init = Div.va.implicit_prior_sampler(n_samples_prior).numpy()


num_epochs = 7000
loss_fct = "LB_MI"
optimizer = torch_Adam
num_samples_MI = 200
freq_MI = 100
save_best_param = True
learning_rate = 0.025

if not load_results_Normal_nocstr :
    ### Training loop
    MI, range_MI, lower_MI, upper_MI = Div.Partial_autograd(J, N, num_epochs, loss_fct, optimizer,
                                                            num_samples_MI, freq_MI, save_best_par
                                                            learning_rate, momentum=True)


    theta_sample = Div.va.implicit_prior_sampler(n_samples_prior)
    with torch.no_grad():
        theta_sample = theta_sample.numpy()

    seed_all(0)

    # Sample data from 'true' parameter
    theta_true = torch.tensor(1.)
    N = 10
    D = Normal.sample(theta_true, N, 1)
    X = D[:, 0]
    # Posterior samples using Jeffreys prior
    n_samples_post = 5*10**4
    jeffreys_post = Normal.sample_post_Jeffreys(X, n_samples_post)
```

```
    jeffreys_post = np.reshape(jeffreys_post, (n_samples_post,q))

    T_mcmc = 10**5 + 1
    sigma2_0 = torch.tensor(1.)
    eps_0 = torch.randn(p)
    eps_MH, batch_acc = VA.MH_posterior(eps_0, T_mcmc, sigma2_0, adap=True, Cov=False)
    theta_MH = NN(eps_MH)

    with torch.no_grad():
        theta_MH = theta_MH.numpy()
        jeffreys_post = jeffreys_post.numpy()
    theta_post = theta_MH[-n_samples_post:]
    theta_post = np.reshape(theta_post, (n_samples_post,q))

    # Saves all relevant quantities

    Mutual_Info = {'values' : MI, 'range' : range_MI, 'lower' : lower_MI, 'upper' : upper_MI}
    Prior = {'samples' : theta_sample, 'jeffreys' : jeffreys_sample, 'params' : all_params}
    Posterior = {'samples' : theta_post, 'jeffreys' : jeffreys_post, 'Markov' : theta_MH}
    Normal_results_nocstr = {'MI' : Mutual_Info, 'prior' : Prior, 'post' : Posterior}
    with open(file_path, 'wb') as file:
        pickle.dump(Normal_results_nocstr, file)

else :
    with open(file_path, 'rb') as file:
        res = pickle.load(file)
        MI_dic = res['MI']
        Prior = res['prior']
        Posterior = res['post']
        MI, range_MI, lower_MI, upper_MI = MI_dic['values'], MI_dic['range'], MI_dic['lower'], MI_
        theta_sample, jeffreys_sample, all_params = Prior['samples'], Prior['jeffreys'], Prior['pa
        theta_post, jeffreys_post, theta_MH = Posterior['samples'], Posterior['jeffreys'], Posteri
fig, axs = plt.subplots(1, 2, figsize=(12, 3))
axs[0].plot(range_MI, MI, '-', color=rougeCEA)
axs[0].plot(range_MI, MI, '*', color=rougeCEA)
axs[0].plot(range_MI, upper_MI, '-', color='lightgrey')
axs[0].plot(range_MI, lower_MI, '-', color='lightgrey')
axs[0].fill_between(range_MI, lower_MI, upper_MI, color='lightgrey', alpha=0.5)
axs[0].set_xlabel(r"Epochs")
axs[0].set_ylabel(r"Generalized mutual information")
axs[0].grid()
axs[1].hist(theta_sample_init, density=True, bins="rice", color="red", label=r"Initial prior", alp
axs[1].hist(theta_sample, density=True, bins="rice", label=r"Fitted prior", alpha=0.4)
axs[1].set_xlabel(r"$\theta$")
axs[1].legend(fontsize=12)
axs[1].grid()
#plt.tight_layout()
plt.show()
```

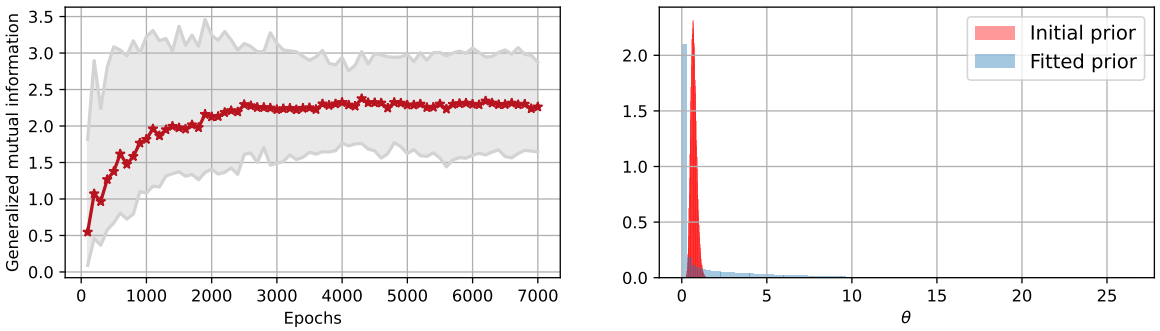

Figure 10: Left : Monte Carlo estimation of the generalized mutual information with $\alpha = 0.5$ (from 200 samples) for π_{λ_e} where λ_e is the parameter of the neural network at epoch e. The red curve is the mean value and the gray zone is the 95% confidence interval. Right : Histograms of the initial prior (at epoch 0) and the fitted prior (after training), each one is obtained from 10^5 samples. The learning rate used in the optimization is 0.025.

We retrieve close results to those of Gauchy et al. (2023), even though we used the α-divergence instead of the classic KL-divergence (Figure 10). The evolution of the mutual information seems to be more stable during training. We can not however directly compare our result to the target Jeffrey prior since the latter is improper.

For the posterior distribution, we sample 10 times from the normal distribution using $\theta_{true} = 1$.

```
fig, axs = plt.subplots(1, 2, figsize=(12, 3))
axs[0].plot(theta_MH)
axs[0].set_xlabel('Iterations')
axs[0].set_ylabel(r'$\theta$')
axs[0].grid()
axs[1].hist(theta_post, density=True, bins="rice", label=r"Fitted posterior", alpha=0.4)
axs[1].hist(jeffreys_post, density=True, bins="rice", label=r"Jeffreys", alpha=0.4, color=vertCEA)
axs[1].set_xlabel(r"$\theta$")
axs[1].legend(fontsize=12)
axs[1].grid()
plt.show()
```

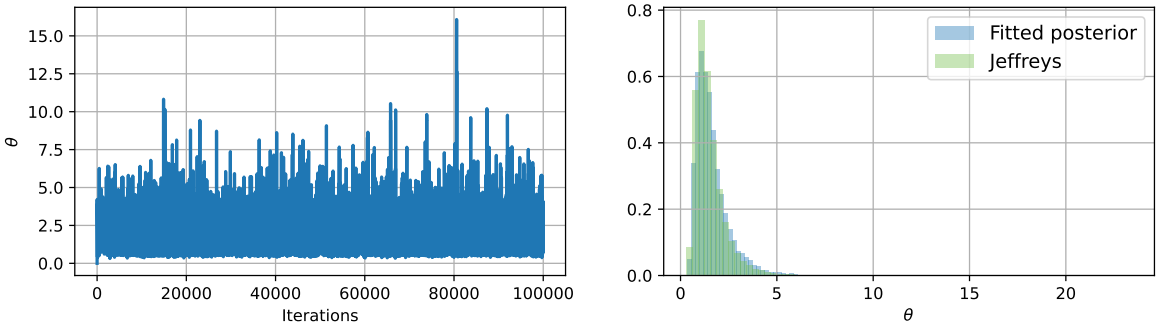

Figure 11: Left : Markov chain during the Metropolis-Hastings iterations. Right : Histograms of the fitted posterior and the Jeffreys posterior, each one is obtained from $5 \cdot 10^4$ samples.

As Figure 11 shows, we obtain a parametric posterior distribution which closely resembles the target distribution, even though the theoretical prior is improper.

In order to evaluate the performance of the algorithm for the prior, we have to add a constraint. The simplest kind of constraints are moment constraints with : $a(\theta) = \theta^{\beta}$, however, we can not use such a constraint here since the integrals for $\mathscr{K}$ and c from section 2 would diverge either at 0 or at $+\infty$.

If we define : $a(\theta) = \dfrac{1}{\theta^{\beta} + \theta^{\tau}}$ with $\beta < 0 < \tau$, then the integrals for $\mathscr{K}$ and c are finite, because :

$$\forall\, \delta \geq 1, \quad \int_0^{+\infty} \frac{1}{\theta} \cdot \left(\frac{1}{\theta^{\beta} + \theta^{\tau}} \right)^{\delta} d\theta \leq \frac{1}{\delta} \left(\frac{1}{\tau} - \frac{1}{\beta} \right).$$

This function of constraint a is preferable because it yields different asymptotic rates at 0 and $+\infty$:

$$\begin{cases} a(\theta) \sim \theta^{-\beta} & \text{as} \quad \theta \longrightarrow 0 \\ a(\theta) \sim \theta^{-\tau} & \text{as} \quad \theta \longrightarrow +\infty. \end{cases}$$

In order to apply the algorithm, we are interested in finding :

$$\mathscr{K} = \int_0^{+\infty} \frac{1}{\theta} \cdot a(\theta)^{1/\alpha} d\theta \quad \text{and} \quad c = \int_0^{+\infty} \frac{1}{\theta} \cdot a(\theta)^{1+(1/\alpha)} d\theta.$$

For instance, let $\alpha = 1/2$. If $\beta = -1$, $\tau = 1$, then $\mathscr{K} = 1/2$ and $c = \pi/16$. The constraint value is $c/\mathscr{K} = \pi/8$. Thus, for this example, we only have to apply the third step of the proposed method. We use in this case a one-layer neural network with exp as the activation function, the parametric set of priors corresponds to log-normal distributions.

```
# Parameters and classes
p = 10      # latent space dimension
q = 1       # theta dimension
mu = 0      # normal mean parameter
N = 10      # number of data samples
J = 1000    # nb samples for MC estimation in MI
T = 50      # nb samples MC marginal likelihood
alpha = 0.5
Normal = torch_NormalModel_variance(mu=mu)
input_size = p
output_size = 1
low = 0.0001        # lower bound
upp = 1 + low       # upper bound (not relevant)
n_samples_prior = 10**6
name_file = 'Normal_results_constrained.pkl'
file_path = os.path.join(path_plot, name_file)

# Constraint function parameter
beta = torch.tensor([-1])
tau = torch.tensor([1])
# beta = -tau = -1, yields K = 0.5
# beta = -tau = -1/3, yields K = 1.5
K = 0.5
constr_val = np.pi / 8
b = torch.tensor([[constr_val]])
T_cstr = 100000
```

```
eta_augm = torch.tensor([[1.0]])
eta = torch.tensor([[1.]])

seed_all(0)
NN = SingleLinear(input_size, output_size, m1=0, s1=0.1, b1=0, act1=torch.exp)
NN = AffineTransformation(NN, m_low=low, M_upp=upp)
VA = VA_NeuralNet(neural_net=NN, model=Normal)
#print(f'Number of parameters : {VA.nb_param}')
#Div = DivMetric_NeuralNet(va=VA, T=T, use_alpha=False, use_log_lik=True)
Div = DivMetric_NeuralNet(va=VA, T=T, use_alpha=True, alpha=alpha, use_log_lik=True)
Constr = Constraints_NeuralNet(div=Div, betas=beta, b=b, T_cstr=T_cstr,
                               objective='LB_MI', lag_method='augmented', eta_augm=eta_augm, rule=

num_epochs = 10000
optimizer = torch_Adam
num_samples_MI = 100
freq_MI = 200
save_best_param = False
learning_rate = 0.0005
freq_augm = 100

def target_prior(theta, beta, tau, K, alpha):
    return (K**-1 / theta) * (theta**beta + theta**tau)**(-1/alpha)

if not load_results_Normal_cstr :

    ### Training loop
    MI, constr_values, range_MI, lower_MI, upper_MI = Constr.Partial_autograd(J, N, eta, num_epochs
                                                      freq_MI, save_best_param, learning_rat
                                                      freq_augm=freq_augm, max_violation=0.0

    with torch.no_grad():
        theta_sample = Constr.va.implicit_prior_sampler(n_samples_prior).numpy()
    print(f'Moment estimation : {np.mean((theta_sample**beta.item()+theta_sample**tau.item())**-1)
    all_params = torch.cat([param.view(-1) for param in NN.parameters()])

    seed_all(0)

    # Sample data from 'true' parameter
    theta_true = torch.tensor(1.)
    N = 10
    D = Normal.sample(theta_true, N, 1)
    X = D[:, 0]

    a_X = 0.5 * N
    b_X = 0.5 * np.sum(X.numpy()**2)
    Y = scipy.stats.invgamma.rvs(a_X, 0, b_X, 10**6)
    cste_post = np.mean((Y**beta.item() + Y**tau.item())**(-1/alpha))
    print(f'Normalizing constant : {cste_post}')
```

```
    def target_posterior(theta, beta, tau, alpha, cste):
        return scipy.stats.invgamma.pdf(theta, a_X, 0, b_X) * (theta**beta + theta**tau)**(-1/alpha

    interv = np.linspace(0.01,10,10000)
    target_post_values = target_posterior(interv,beta.item(),tau.item(),alpha,cste_post)

    n_samples_post = 5*10**4
    T_mcmc = 10**5 + 1
    sigma2_0 = torch.tensor(1.)
    eps_0 = torch.randn(p)
    eps_MH, batch_acc = VA.MH_posterior(eps_0, T_mcmc, sigma2_0, adap=True, Cov=False)
    theta_MH = NN(eps_MH)

    with torch.no_grad():
        theta_MH = theta_MH.numpy()
    theta_post = theta_MH[-n_samples_post:]
    theta_post = np.reshape(theta_post, n_samples_post)
    # Saves all relevant quantities

    Mutual_Info = {'values' : MI, 'range' : range_MI, 'lower' : lower_MI, 'upper' : upper_MI}
    Prior = {'samples' : theta_sample, 'jeffreys' : jeffreys_sample, 'params' : all_params}
    Posterior = {'samples' : theta_post, 'jeffreys' : jeffreys_post, 'target' : target_post_values}
    Normal_results_cstr = {'MI' : Mutual_Info, 'prior' : Prior, 'post' : Posterior}
    with open(file_path, 'wb') as file:
        pickle.dump(Normal_results_cstr, file)

else :
    with open(file_path, 'rb') as file:
        res = pickle.load(file)
        MI_dic = res['MI']
        Prior = res['prior']
        Posterior = res['post']
        MI, range_MI, lower_MI, upper_MI = MI_dic['values'], MI_dic['range'], MI_dic['lower'], MI_
        theta_sample, jeffreys_sample, all_params = Prior['samples'], Prior['jeffreys'], Prior['pa
        theta_post, jeffreys_post, target_post_values = Posterior['samples'], Posterior['jeffreys']

interv = np.linspace(0.01,10,10000)
plt.figure(figsize=(5, 3))
plt.hist(theta_sample, density=True, bins=500, label=r"Fitted prior", alpha=0.4)
plt.plot(interv, target_prior(interv,beta.item(),tau.item(),K,alpha), label="Target prior",color=v
plt.xlabel(r"$\theta$")
plt.legend(fontsize=12)
plt.grid()
plt.xlim(-0.1,10)
plt.tight_layout()
```

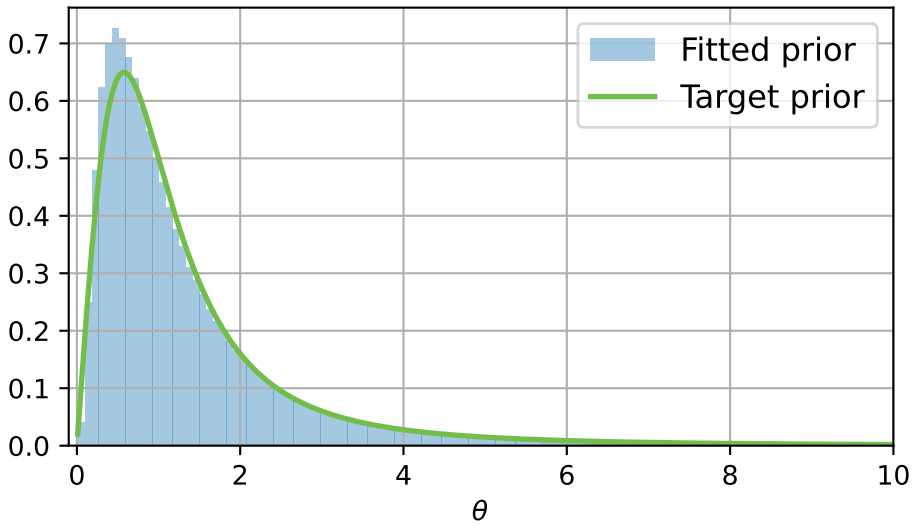

Figure 12: Histogram of the constrained fitted prior obtained from 10^5 samples, and density function of the target prior. The learning rate used in the optimization is 0.0005.

In this case we are able to compare prior distributions since both are proper, as Figure 12 shows, we recover a relevant result using our algorithm even with added constraints.

The density function of the posterior is known up to a multiplicative constant, more precisely, it corresponds to the product of the constraint function and the density function of an inverse-gamma distribution. Hence, the constant can be estimated using Monte Carlo samples from the inverse-gamma distribution in question. We apply the same approach as before in order to obtain the posterior from the parametric prior.

```
interv = np.linspace(0.01,10,10000)
plt.figure(figsize=(5, 3))
plt.hist(theta_post, density=True, bins=500, label=r"Fitted posterior", alpha=0.4)
plt.plot(interv, target_post_values, label="Target posterior",color=vertCEA)
plt.xlabel(r"$\theta$")
plt.legend(fontsize=12)
plt.grid()
plt.xlim(-0.1,6)
plt.tight_layout()
```

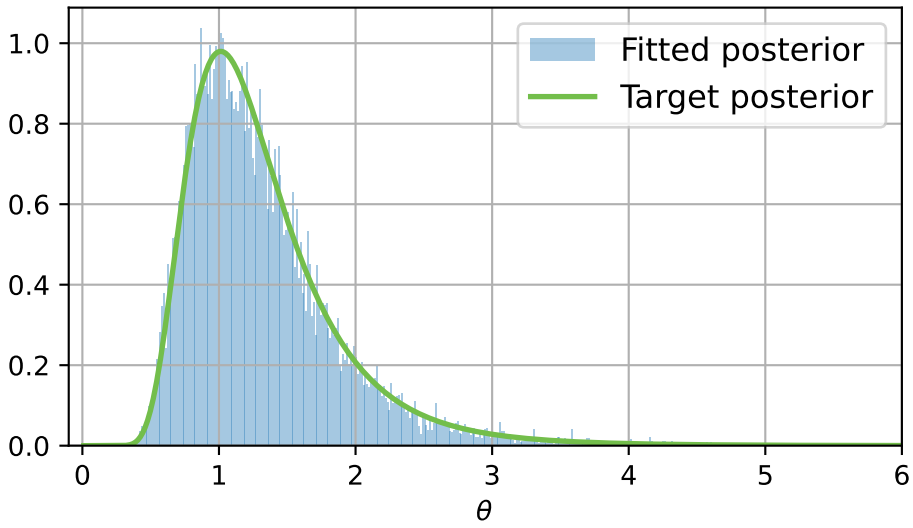

Figure 13: Histogram of the fitted posterior obtained from $5 \cdot 10^4$ samples, and density function of the target posterior.

As shown in Figure 13, the parametric posterior has a shape similar to the theoretical distribution.

6.3 Probit model and robustness

As mentioned in Section 4.2 regarding the probit model, we present several additional results.

```
num_M = 10
N_max = 200
N_init = 5
tab_N = np.arange(N_init, N_max+N_init, 5)
QE = np.zeros((len(tab_N), num_M))
file_name = os.path.join(varp_probit_path, 'Quadratic_errors/Quad_error_post')
first_file = file_name + '1.pkl'
for M in range(num_M):
    file = file_name + f'{M+1}.pkl'
    with open(file, 'rb') as file:
        QE[:,M] = pickle.load(file)

mean_vals = np.mean(QE, axis=1)
lower_quantile = np.quantile(QE, 0.025, axis=1)
upper_quantile = np.quantile(QE, 0.975, axis=1)

seed_all(1)
theta_vrai = np.array([3.37610525, 0.43304097])
N_max = 200
N_init = 5
tab_N2 = np.arange(N_init, N_max+N_init, 5)
num_MC = 5000
M = 10
num_MC = 5000
err_jeff = np.zeros((len(tab_N2), M))
```

```
j = 0
for N in tab_N2 :
    resultats_theta_post_N = np.zeros((M, num_MC, 2))
    for i in tqdm(range(M), desc=f'Iterations for N={N}', disable=True):
        file_path = os.path.join(int_jeffreys_path, f'model_J_{i}')
        with open(file_path, 'rb') as file:
            Jeffreys = pickle.load(file)
        resultats_theta_post_N[i] = Jeffreys['logs']['post'][N]
        err_jeff[j, i] = np.mean(np.linalg.norm(resultats_theta_post_N[i] - theta_vrai[np.newaxis]
    j = j + 1

mean_vals2 = np.mean(err_jeff, axis=1)
lower_quantile2 = np.quantile(err_jeff, 0.025, axis=1)
upper_quantile2 = np.quantile(err_jeff, 0.975, axis=1)

plt.figure(figsize=(5, 3))
plt.plot(tab_N, mean_vals, label='Mean Approx')
plt.fill_between(tab_N, lower_quantile, upper_quantile, color='b', alpha=0.2, label='95% CI Approx

plt.plot(tab_N2, mean_vals2, label='Mean Jeffreys')
plt.fill_between(tab_N2, lower_quantile2, upper_quantile2, color='orange', alpha=0.2, label='95% C
plt.xlabel('N')
plt.ylabel('Mean Norm Difference')
plt.legend(fontsize=12)
plt.grid()
plt.show()
```

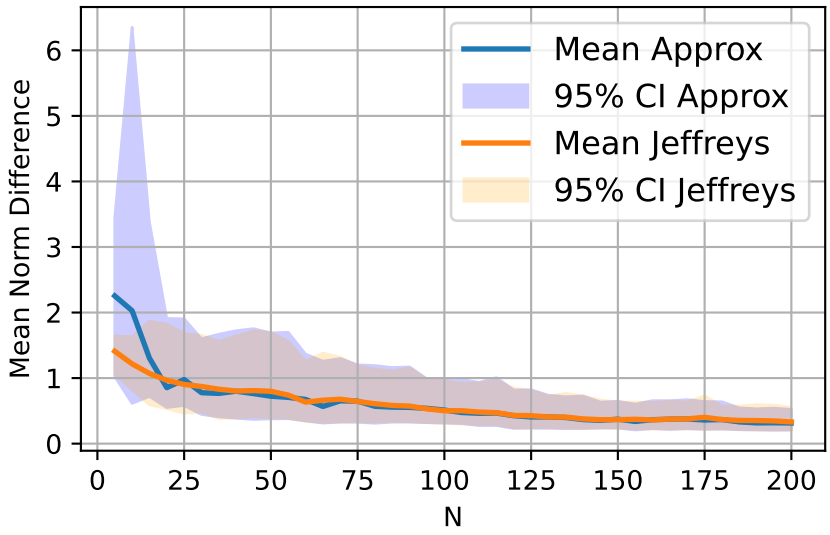

Figure 14: Mean norm difference as a function of the size N of the dataset for the unconstrained fitted posterior and the Jeffreys posterior. For each value of N, 10 different datasets are considered from which we obtain 95% confidence intervals.

```
num_M = 10
N_max = 200
N_init = 5
```

```
tab_N = np.arange(N_init, N_max+N_init, 5)
QE = np.zeros((len(tab_N), num_M))
file_name = os.path.join(varp_probit_path, 'Quadratic_errors/Quad_error_post_constr')
first_file = file_name + '1.pkl'
for M in range(num_M):
    file = file_name + f'{M+1}.pkl'
    with open(file, 'rb') as file:
        QE[:,M] = pickle.load(file)

mean_vals = np.mean(QE, axis=1)
lower_quantile = np.quantile(QE, 0.025, axis=1)
upper_quantile = np.quantile(QE, 0.975, axis=1)

seed_all(1)
theta_vrai = np.array([3.37610525, 0.43304097])
N_max = 200
N_init = 5
tab_N2 = np.arange(N_init, N_max+N_init, 5)
num_MC = 5000
M = 10
num_MC = 5000
err_jeff = np.zeros((len(tab_N2), M))
j = 0
for N in tab_N2 :
    resultats_theta_post_N = np.zeros((M, num_MC, 2))
    for i in tqdm(range(M), desc=f'Iterations for N={N}', disable=True):
        file_path = os.path.join(int_jeffreys_path, f'model_J_constraint_{i}')
        with open(file_path, 'rb') as file:
            Jeffreys = pickle.load(file)
        resultats_theta_post_N[i] = Jeffreys['logs']['post'][N]
        err_jeff[j, i] = np.mean(np.linalg.norm(resultats_theta_post_N[i] - theta_vrai[np.newaxis]
    j = j + 1

mean_vals2 = np.mean(err_jeff, axis=1)
lower_quantile2 = np.quantile(err_jeff, 0.025, axis=1)
upper_quantile2 = np.quantile(err_jeff, 0.975, axis=1)

plt.figure(figsize=(5, 3))
plt.plot(tab_N, mean_vals, label='Mean Approx')
plt.fill_between(tab_N, lower_quantile, upper_quantile, color='b', alpha=0.2, label='95% CI Approx

plt.plot(tab_N2, mean_vals2, label='Mean Jeffreys')
plt.fill_between(tab_N2, lower_quantile2, upper_quantile2, color='orange', alpha=0.2, label='95% C
plt.xlabel('N')
plt.ylabel('Mean Norm Difference')
plt.legend(fontsize=12)
plt.grid()
plt.show()
```

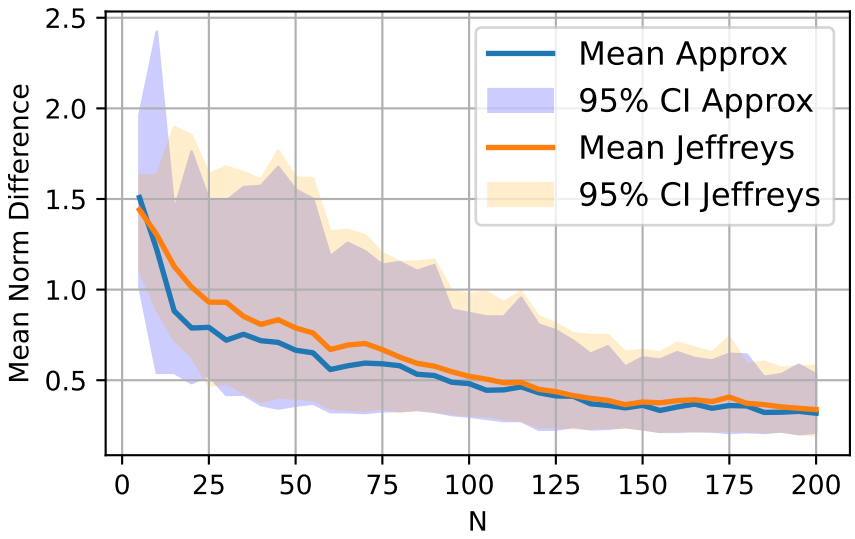

Figure 15: Mean norm difference as a function of the size N of the dataset for the constrained fitted posterior and the Jeffreys posterior. For each value of N, 10 different datasets are considered from which we obtain 95% confidence intervals.

Figure 14 and Figure 15 show the evolution of the posterior mean norm difference as the size N of the dataset considered for the posterior distribution increases. For each value of N, 10 different datasets are used in order to quantify the variability of said error. The proportion of degenerate datasets is rather high when $N = 5$ or $N = 10$, the consequence is that the approximation tends to be more unstable. The main observation is that the error is decreasing in all cases when N increases, also, the behaviour of the error for the fitted distributions on one hand, and the behaviour for the Jeffreys distribution on the other hand are quite similar in terms of mean value and confidence intervals.

```
n_exp = 100   # Number of experiments
model = os.path.join(varp_probit_path, 'probit_samples/probit')
q = 2
first_file = model + '_seed_1.pkl'

n_samples_prior = 10**6
n_samples_post = 5000

# Concatenate results from all experiments

theta_prior = np.zeros((n_exp,n_samples_prior,q))
jeffreys_prior = np.zeros((n_exp,n_samples_prior,q))
theta_post = np.zeros((n_exp,n_samples_post,q))
jeffreys_post = np.zeros((n_exp,n_samples_post,q))

for k in range(n_exp):
    file_name = model + f'_seed{k+1}.pkl'
    with open(file_name, 'rb') as file:
        data = pickle.load(file)
        theta_prior[k,:,:] = data['prior']['VA']
        #jeffreys_prior[k,:,:] = data['prior']['Jeffreys']
        theta_post[k,:,:] = data['post']['VA']
```

```
        jeffreys_post[k,:,:] = data['post']['Jeffreys']

# Post CDF computations
num_x = 5000
x_values_alpha = np.linspace(0., 8., num_x)
x_values_beta = np.linspace(0., 2., num_x)
cdf_post_alpha = np.zeros((n_exp, num_x))
cdf_post_beta = np.zeros((n_exp, num_x))
jeff_post_alpha = np.zeros((n_exp, num_x))
jeff_post_beta = np.zeros((n_exp, num_x))
for k in range(n_exp):
    cdf_post_alpha[k,:] = compute_ecdf(theta_post[k,:,0], x_values_alpha)
    cdf_post_beta[k,:] = compute_ecdf(theta_post[k,:,1], x_values_beta)
    jeff_post_alpha[k,:] = compute_ecdf(jeffreys_post[k,:,0], x_values_alpha)
    jeff_post_beta[k,:] = compute_ecdf(jeffreys_post[k,:,1], x_values_beta)


lower_bound_alpha = np.quantile(cdf_post_alpha, 0., axis=0)
upper_bound_alpha = np.quantile(cdf_post_alpha, 1.0, axis=0)
median_cdf_alpha = np.quantile(cdf_post_alpha, 0.5, axis=0)
median_jeff_alpha = np.quantile(jeff_post_alpha, 0.5, axis=0)

lower_bound_beta = np.quantile(cdf_post_beta, 0., axis=0)
upper_bound_beta = np.quantile(cdf_post_beta, 1., axis=0)
median_cdf_beta = np.quantile(cdf_post_beta, 0.5, axis=0)
median_jeff_beta = np.quantile(jeff_post_beta, 0.5, axis=0)

# Create subplots for alpha and beta
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 3))

# Plot for alpha
ax1.plot(x_values_alpha, median_cdf_alpha, label='Median fitted ECDF', color='blue', linewidth=2)
ax1.plot(x_values_alpha, median_jeff_alpha, label='Jeffreys ECDF', color='green', linewidth=2)
ax1.fill_between(x_values_alpha, lower_bound_alpha, upper_bound_alpha, color='lightblue', alpha=0.
ax1.set_xlabel(r'$\theta_1$')
ax1.set_ylabel('CDF')
ax1.grid()
ax1.legend(loc='lower right', fontsize=12)

# Plot for beta
ax2.plot(x_values_beta, median_cdf_beta, label='Median fitted ECDF', color='blue', linewidth=2)
ax2.plot(x_values_beta, median_jeff_beta, label='Jeffreys ECDF', color='green', linewidth=2)
ax2.fill_between(x_values_beta, lower_bound_beta, upper_bound_beta, color='lightblue', alpha=0.5,
ax2.set_xlabel(r'$\theta_2$')
ax2.set_ylabel('CDF')
ax2.grid()
ax2.legend(loc='lower right', fontsize=12)
plt.tight_layout()
plt.show()
```

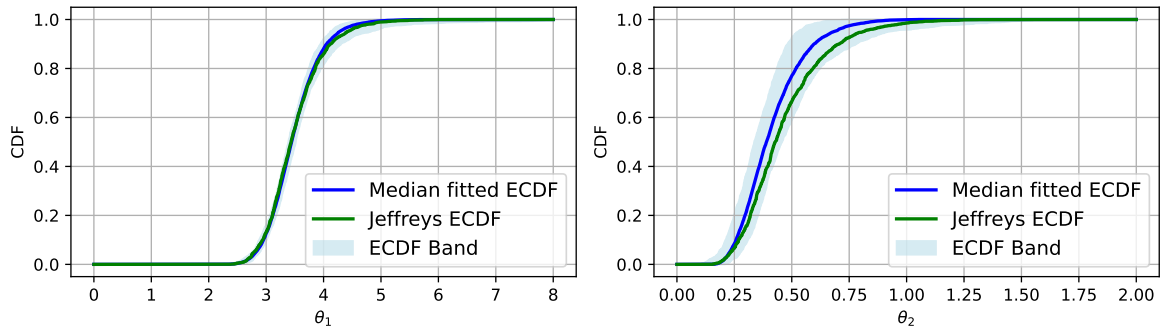

Figure 16: Empirical cumulative distribution functions for the unconstrained fitted posterior and the Jeffreys posterior using 5000 samples. The band is obtained by computing the ECDFs over 100 different seeds and monitoring the maximum and minimum ECDF values for each θ.

```
n_exp = 100   # Number of experiments
model = os.path.join(varp_probit_path, 'probit_samples/probit_constr')
q = 2
first_file = model + '_seed1.pkl'
n_samples_prior = 10**6
n_samples_post = 5000

# Concatenate results from all experiments

theta_prior = np.zeros((n_exp,n_samples_prior,q))
jeffreys_prior = np.zeros((n_exp,n_samples_prior,q))
theta_post = np.zeros((n_exp,n_samples_post,q))
jeffreys_post = np.zeros((n_exp,n_samples_post,q))

for k in range(n_exp):
    file_name = model + f'_seed{k+1}.pkl'
    #file_name = model + f'_seed_lognormal{k+1}.pkl'
    with open(file_name, 'rb') as file:
        data = pickle.load(file)
        theta_prior[k,:,:] = data['prior']['VA']
        #jeffreys_prior[k,:,:] = data['prior']['Jeffreys']
        theta_post[k,:,:] = data['post']['VA']
        jeffreys_post[k,:,:] = data['post']['Jeffreys']

# Post CDF computations
num_x = 5000
x_values_alpha = np.linspace(0., 8., num_x)
x_values_beta = np.linspace(0., 2., num_x)
cdf_post_alpha = np.zeros((n_exp, num_x))
cdf_post_beta = np.zeros((n_exp, num_x))
jeff_post_alpha = np.zeros((n_exp, num_x))
jeff_post_beta = np.zeros((n_exp, num_x))
for k in range(n_exp):
    cdf_post_alpha[k,:] = compute_ecdf(theta_post[k,:,0], x_values_alpha)
    cdf_post_beta[k,:] = compute_ecdf(theta_post[k,:,1], x_values_beta)
    jeff_post_alpha[k,:] = compute_ecdf(jeffreys_post[k,:,0], x_values_alpha)
```

```
    jeff_post_beta[k,:] = compute_ecdf(jeffreys_post[k,:,1], x_values_beta)


lower_bound_alpha = np.quantile(cdf_post_alpha, 0., axis=0)
upper_bound_alpha = np.quantile(cdf_post_alpha, 1.0, axis=0)
median_cdf_alpha = np.quantile(cdf_post_alpha, 0.5, axis=0)
median_jeff_alpha = np.quantile(jeff_post_alpha, 0.5, axis=0)

lower_bound_beta = np.quantile(cdf_post_beta, 0., axis=0)
upper_bound_beta = np.quantile(cdf_post_beta, 1.0, axis=0)
median_cdf_beta = np.quantile(cdf_post_beta, 0.5, axis=0)
median_jeff_beta = np.quantile(jeff_post_beta, 0.5, axis=0)

# Create subplots for alpha and beta
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 3))

# Plot for alpha
ax1.plot(x_values_alpha, median_cdf_alpha, label='Median fitted ECDF', color='blue', linewidth=2)
ax1.plot(x_values_alpha, median_jeff_alpha, label='Jeffreys ECDF', color='green', linewidth=2)
ax1.fill_between(x_values_alpha, lower_bound_alpha, upper_bound_alpha, color='lightblue', alpha=0.
ax1.set_xlabel(r'$\theta_1$')
ax1.set_ylabel('CDF')
ax1.grid()
ax1.legend(loc='lower right', fontsize=12)

# Plot for beta
ax2.plot(x_values_beta, median_cdf_beta, label='Median fitted ECDF', color='blue', linewidth=2)
ax2.plot(x_values_beta, median_jeff_beta, label='Jeffreys ECDF', color='green', linewidth=2)
ax2.fill_between(x_values_beta, lower_bound_beta, upper_bound_beta, color='lightblue', alpha=0.5,
ax2.set_xlabel(r'$\theta_2$')
ax2.set_ylabel('CDF')
ax2.grid()
ax2.legend(loc='lower right', fontsize=12)
plt.tight_layout()
plt.show()
```

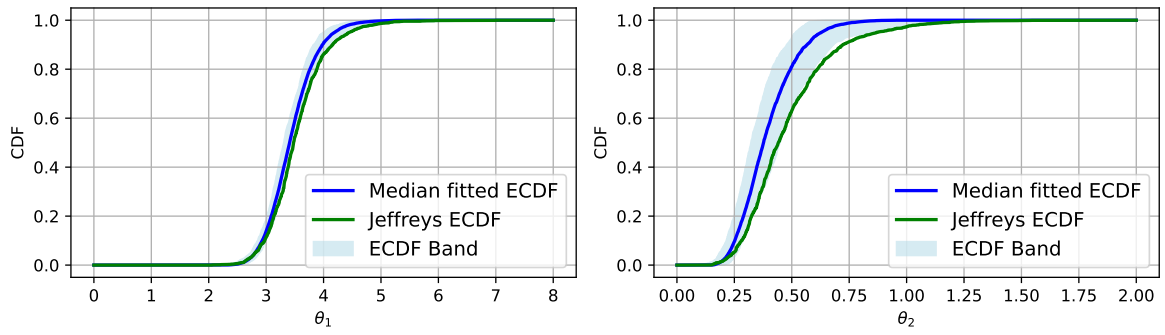

Figure 17: Empirical cumulative distribution functions for the constrained fitted posterior and the Jeffreys posterior using 5000 samples. The band is obtained by computing the ECDFs over 100 different seeds and monitoring the maximum and minimum ECDF values for each θ.

Figure 16 and Figure 17 compare the empirical distribution functions of the fitted posterior and the Jeffreys posterior. In the unconstrained case, one can observe that the ECDFs are very close for θ_1, whereas the variability is slightly higher for θ_2 although still reasonable. When imposing a constraint on θ_2, one remarks that the variability of the result is higher. The Jeffreys ECDF is contained in the band when θ_2 is close to zero, but not when θ_2 increases ($\theta_2 > 0.5$). This is coherent with the previous scatter histograms where the Jeffreys posterior on θ_2 tends to have a heavier tail than the variational approximation.

Altogether, despite the stochastic nature of the developed algorithm, we consider that the result tends to be reasonably robust to the RNG seed for the optimization part, and robust to the dataset used for the posterior distribution for the MCMC part.

References

Basir, Shamsulhaq, and Inanc Senocak. 2023. "An Adaptive Augmented Lagrangian Method for Training Physics and Equality Constrained Artificial Neural Networks." https://arxiv.org/abs/2306.04904.

Berger, James O., José M. Bernardo, and Dongchu Sun. 2009. "The formal definition of reference priors." *The Annals of Statistics* 37 (2): 905–38. https://doi.org/10.1214/07-AOS587.

Bernardo, José M. 1979a. "Expected Information as Expected Utility." *The Annals of Statistics* 7 (3): 686–90. https://doi.org/10.1214/aos/1176344689.

———. 1979b. "Reference Posterior Distributions for Bayesian Inference." *Journal of the Royal Statistical Society. Series B* 41 (2): 113–47. https://doi.org/10.1111/j.2517-6161.1979.tb01066.x.

———. 2005. "Reference Analysis." In *Bayesian Thinking*, edited by D. K. Dey and C. R. Rao, 25:17–90. Handbook of Statistics. Elsevier. https://doi.org/10.1016/S0169-7161(05)25002-2.

Bioche, Christele, and Pierre Druilhet. 2016. "Approximation of Improper Priors." *Bernoulli* 22 (3): 1709–28. https://doi.org/10.3150/15-bej708.

Bousquet, Nicolas. 2008. "Eliciting Vague but Proper Maximal Entropy Priors in Bayesian Experiments." *Statistical Papers* 51 (3): 613–28. https://doi.org/10.1007/s00362-008-0149-9.

Clarke, Bertrand S., and Andrew R. Barron. 1994. "Jeffreys' Prior Is Asymptotically Least Favorable Under Entropy Risk." *Journal of Statistical Planning and Inference* 41 (1): 37–60. https://doi.org/10.1016/0378-3758(94)90153-8.

D'Andrea, Vera L. D. AND Aljohani, Amanda M. E. AND Tomazella. 2021. "Objective Bayesian Analysis for Multiple Repairable Systems." *PLOS ONE* 16 (November): 1–19. https://doi.org/10.1371/journal.pone.0258581.

Gao, Yansong, Rahul Ramesh, and Pratik Chaudhari. 2022. "Deep Reference Priors: What Is the Best Way to Pretrain a Model?" In *Proceedings of the 39th International Conference on Machine Learning*, 162:7036–51. Proceedings of Machine Learning Research. PMLR. https://Proceedings.mlr.press/v162/gao22d.html.

Gauchy, Clément. 2022. "Uncertainty quantification methodology for seismic fragility curves of mechanical structures : Application to a piping system of a nuclear power plant." Theses, Institut Polytechnique de Paris. https://theses.hal.science/tel-04102809.

Gauchy, Clément, Antoine Van Biesbroeck, Cyril Feau, and Josselin Garnier. 2023. "Inférence Variationnelle de Lois a Priori de Référence." In *Proceedings Des 54èmes Journées de Statistiques (JdS)*. SFDS. https://jds2023.sciencesconf.org/resource/page/id/19.

Gelman, Andrew, John B. Carlin, Hal S. Stern, David B. Dunson, Aki Vehtari, and Donald B. Rubin. 2013. "Bayesian Data Analysis, Third Edition." In, 293–300. Chapman; Hall/CRC. https://doi.org/10.1201/b16018.

Gretton, Arthur, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander Smola. 2012. "A Kernel Two-Sample Test." *Journal of Machine Learning Research* 13 (25): 723–73.

http://jmlr.org/papers/v13/gretton12a.html.
Gu, Mengyang, and James O. Berger. 2016. "Parallel partial Gaussian process emulation for computer models with massive output." *The Annals of Applied Statistics* 10 (3): 1317–47. https://doi.org/10.1214/16-AOAS934.
Jang, Eric, Shixiang Gu, and Ben Poole. 2017. "Categorical Reparameterization with Gumbel-Softmax." In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*. Toulon, France. https://doi.org/10.48550/arXiv.1611.01144.
Jaynes, E. T. 1957. "Information Theory and Statistical Mechanics." *Phys. Rev.* 106 (May): 620–30. https://doi.org/10.1103/PhysRev.106.620.
Kass, Robert E., and Larry Wasserman. 1996. "The Selection of Prior Distributions by Formal Rules." *Journal of the American Statistical Association* 91 (435): 1343–70. https://doi.org/10.1080/01621459.1996.10477003.
Kennedy, Robert P., C. Allin Cornell, Robert D. Campbell, Stan J. Kaplan, and F. Harold. 1980. "Probabilistic Seismic Safety Study of an Existing Nuclear Power Plant." *Nuclear Engineering and Design* 59 (2): 315–38. https://doi.org/10.1016/0029-5493(80)90203-4.
Kingma, Diederik P., and Jimmy Ba. 2015. "Adam: A Method for Stochastic Optimization." In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*. San Diego, USA. https://doi.org/10.48550/arXiv.1412.6980.
Kingma, Diederik P., and Max Welling. 2014. "Auto-Encoding Variational Bayes." In *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*. Banff, Canada. https://doi.org/10.48550/arXiv.1312.6114.
———. 2019. "An Introduction to Variational Autoencoders." *Foundations and Trends® in Machine Learning* 12 (4): 307--392. https://doi.org/10.1561/2200000056.
Kobyzev, Ivan, Simon J. D. Prince, and Marcus A. Brubaker. 2021. "Normalizing Flows: An Introduction and Review of Current Methods." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43 (11): 3964–79. https://doi.org/10.1109/TPAMI.2020.2992934.
Lafferty, John D., and Larry A. Wasserman. 2001. "Iterative Markov Chain Monte Carlo Computation of Reference Priors and Minimax Risk." In *Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence (UAI)*, edited by Jack S. Breese and Daphne Koller, 293–300. Seattle, USA: Morgan Kaufmann. https://doi.org/10.48550/arXiv.1301.2286.
Li, Hanmo, and Mengyang Gu. 2021. "Robust Estimation of SARS-CoV-2 Epidemic in US Counties." *Scientific Reports* 11 (11841): 2045–2322. https://doi.org/10.1038/s41598-021-90195-6.
MacKay, D. J. C. 2003. *Information Theory, Inference, and Learning Algorithms*. Copyright Cambridge University Press.
Marzouk, Y., T. Moselhy, M. Parno, and A. Spantini. 2016. "Sampling via Measure Transport: An Introduction." In *Handbook of Uncertainty Quantification*, 1–41. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-11259-6/_23-1.
Muré, Joseph. 2018. "Objective Bayesian Analysis of Kriging Models with Anisotropic Correlation Kernel." PhD thesis, Université Sorbonne Paris Cité. https://theses.hal.science/tel-02184403/file/MURE_Joseph_2_complete_20181005.pdf.
Nalisnick, Eric, and Padhraic Smyth. 2017. "Learning Approximately Objective Priors." In *Proceedings of the 33rd Conference on Uncertainty in Artificial Intelligence (UAI)*. Sydney, Australia: Association for Uncertainty in Artificial Intelligence (AUAI). https://doi.org/10.48550/arXiv.1704.01168.
Natarajan, Ranjini, and Robert E. Kass. 2000. "Reference Bayesian Methods for Generalized Linear Mixed Models." *Journal of the American Statistical Association* 95 (449): 227–37. https://doi.org/10.1080/01621459.2000.10473916.
Nocedal, Jorge, and Stephen J. Wright. 2006. "Numerical Optimization." In *Springer Series in Operations Research and Financial Engineering*, 497–528. Springer New York. https://doi.org/10.1007/978-0-387-40065-5/_17.
Papamakarios, George, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Laksh-

minarayanan. 2021. “Normalizing Flows for Probabilistic Modeling and Inference.” *Journal of Machine Learning Research* 22 (57): 1–64. http://jmlr.org/papers/v22/19-1028.html.

Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, et al. 2019. “PyTorch: An Imperative Style, High-Performance Deep Learning Library.” In *Advances in Neural Information Processing Systems*, edited by H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett. Vol. 32. Curran Associates, Inc. https://Proceedings.neurips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf.

Paulo, Rui. 2005. “Default priors for Gaussian processes.” *The Annals of Statistics* 33 (2): 556–82. https://doi.org/10.1214/009053604000001264.

Press, S James. 2009. *Subjective and Objective Bayesian Statistics: Principles, Models, and Applications*. John Wiley & Sons.

Reid, N, R Mukerjee, and DAS Fraser. 2003. “Some Aspects of Matching Priors.” *Lecture Notes-Monograph Series*, 31–43.

Simpson, Daniel, Håvard Rue, Andrea Riebler, Thiago G. Martins, and Sigrunn H. Sørbye. 2017. “Penalising Model Component Complexity: A Principled, Practical Approach to Constructing Priors.” *Statistical Science* 32 (1): 1–28. https://doi.org/10.1214/16-STS576.

Soofi, Ehsan S. 2000. “Principal Information Theoretic Approaches.” *Journal of the American Statistical Association* 95 (452): 1349–53. https://doi.org/10.1080/01621459.2000.10474346.

Van Biesbroeck, Antoine. 2024a. “Generalized Mutual Information and Their Reference Priors Under Csizar f-Divergence.” https://arxiv.org/abs/2310.10530.

———. 2024b. “Properly Constrained Reference Priors Decay Rates for Efficient and Robust Posterior Inference.” https://arxiv.org/abs/2409.13041.

Van Biesbroeck, Antoine, Clément Gauchy, Cyril Feau, and Josselin Garnier. 2024. “Reference Prior for Bayesian Estimation of Seismic Fragility Curves.” *Probabilistic Engineering Mechanics* 76 (April): 103622. https://doi.org/10.1016/j.probengmech.2024.103622.

Zellner, Arnold. 1996. “Models, Prior Information, and Bayesian Analysis.” *Journal of Econometrics* 75 (1): 51–68. https://doi.org/10.1016/0304-4076(95)01768-2.