University of Oxford

Variational Inference with general families of divergence measures

by

Stan Koobs

Magdalen College

A dissertation submitted in partial fulfilment of the degree of Master of Science in Statistical Science.

Department of Statistics, 24–29 St Giles,
Oxford, OX1 3LB

September 2022

This is my own work (except where otherwise indicated)

Candidate: Stan Koobs

Signed: 

Date: 11-09-2022

Abstract

Variational Inference is a promising technique that allows us to perform approximate Bayesian inference. It has been shown empirically that Variational Inference can outperform other popular techniques such as **M**arkov **C**hain **M**onte **C**arlo (MCMC), especially in high-dimensional settings. The paper by Daudel et al. (2022) introduced a novel family of iterative algorithms which carry out α-divergence minimisation while using a mixture model as the variational density. In this dissertation, we establish further theoretical justification for this novel approach. Namely, under Gaussianity, we prove convergence results which we later generalise to the exponential family. Moreover, we provide empirical evidence of the derived results. Lastly, we enhance the algorithm by implementing covariance updates and alternative learning rate policies leading to improved empirical performances.

Acknowledgements

First and foremost, I would like to thank my supervisors Dr. Kamélia Daudel and Professor Arnaud Doucet, for their exceptional guidance, very helpful suggestions, and generous investment in their time and support throughout this project. I would also like to express my gratitude to my parents, whose consistent support and encouragement were the reason I was able to study in Oxford this year.

Contents

List of Figures

List of Tables

1

Introduction

The emergence of Big Data brings a lot of opportunities and challenges to the modern world. On the one hand, it gives fields like Machine Learning the possibility to discover small patterns that were not detectable in small-scale data. On the other hand, this tremendous amount of data brings a set of unique statistical problems. In such a complex high-dimensional context, we desire methods that are able to capture potentially complicated structures inside the data. However, these sophisticated models often involve intractable quantities.

This is particularly true in Bayesian Inference. When we are treating the problem in a Bayesian manner, we can incorporate prior beliefs over all parameters and update these to obtain posterior beliefs based on the observed data. By that, it offers us a systematic way to incorporate probabilistic uncertainty in our models. However, the calculations involved with many inferences tasks - prediction, estimation, hypothesis testing - are usually integrations. Especially in these complex high-dimensional models, these integrals are intractable. Therefore the resulting posterior distributions and the corresponding quantities of interests such as the posterior mean, are too expensive to compute.

Approximate Bayesian Inference techniques have been developed to alleviate this burden. Variational Inference is one such technique that allows us to perform approximate inference (Blei et al., 2017; Zhang et al., 2018). It is a popular alternative to the already widely established technique of MCMC. The main benefit of Variational Inference being that it scales easier to high-dimensional problems by relying on the optimisation literature. This is particularly relevant in this Big Data setting where datasets contain millions of parameters. Yet, the empirical performance of Variational Inference methods is often limited because of two factors: *(i)* an inappropriate choice of the objective function appearing in the optimisation problem and *(ii)* a search space that is too restrictive to match the target at the end of the optimisation procedure.

The goal of this dissertation is to explore how one can remedy these two issues by following the approach offered by Daudel et al. (2022). This paper tackles limitation *(i)* by selecting the α-divergence as a more general class of objective functions. Moreover, it enlarges the search space beyond the traditional framework used in Variational Inference by making use mixture models as approximating distribution, thereby dealing with problem *(ii)*. There is empirical evidence that this methodology can enhance existing algorithms in the context of multimodal target distributions.

The remainder of this dissertation is structured as follows. Chapter 2 contains the background knowledge that permits us to introduce the main concepts used in Daudel et al. (2022). Chapter 3 demonstrates our theoretical analysis. In a Gaussian setting, we prove exponential convergence results and provide the optimal learning rate. Subsequently, these results are also generalised to the exponential family. Next to this theoretical work, we perform several numerical experiments which are displayed in Chapter 4. This chapter validates the derived results and scrutinises the performance of the algorithm under several new settings. Finally, Chapter 5 provides concluding remarks and outlines some potential perspectives.

2

Background

The goal of this chapter is to introduce the concepts necessary to comprehend the work done in Daudel et al. (2022), which this dissertation will mainly build upon. We start by reviewing the basics of Bayesian Inference.

2.1 Bayesian Inference

Bayesian Inference is a particular method of statistical inference. As opposed to the frequentist paradigm, in Bayesian Inference we regard unknown variables or parameters as random variables. We perform inference by starting with some prior knowledge and then updating our beliefs given the observed data using Bayes' rule.

We will begin with formulating our Bayesian framework. Let $(\mathsf{Y}, \mathcal{Y}, \nu)$ be a measured space, where ν is a σ-finite measure on the measurable space $(\mathsf{Y}, \mathcal{Y})$. Now we observe some data $\mathcal{D}$ coming from the *likelihood* $p(\mathcal{D}|y)$ which is parameterised by a latent variable $y \in \mathsf{Y}$ drawn from the *prior* $p_0(y)$. Our goal is to draw inference about y after having seen the data $\mathcal{D}$. In Bayesian Inference, we do this by considering the *posterior density* of y given $\mathcal{D}$ which is defined by

$$p(y|\mathcal{D}) = \frac{p(y, \mathcal{D})}{p(\mathcal{D})} = \frac{p_0(y)p(\mathcal{D}|y)}{p(\mathcal{D})}, \tag{2.1.1}$$

where the denominator $p(\mathcal{D}) = \int_{\mathsf{Y}} p(\mathcal{D}|y)p_0(y)\,\nu(\mathrm{d}y)$ is known as the *marginal likelihood* or *(model) evidence*. This posterior density can be utilised to quantify the uncertainty of parameter y after having seen the data $\mathcal{D}$ by analysing quantities such as the *posterior mean*

$$\int_{\mathsf{Y}} y\, p(y|\mathcal{D})\,\nu(\mathrm{d}y).$$

In general, to be able to assess the uncertainty of y, we need to be able to calculate integrals of the form

$$\int_{\mathsf{Y}} g(y)p(y|\mathcal{D})\nu(\mathrm{d}y), \tag{2.1.2}$$

where g specifies the quantity of interest. However, the crucial problem in Bayesian Inference is that there does not exist a general closed-form expression for (2.1.2). One of the reasons for this is that in many applications, the model evidence $p(\mathcal{D})$ is intractable, which means we only know the posterior up to a proportionality constant: $p(y|\mathcal{D}) \propto p(y, \mathcal{D})$. Therefore, exact Bayesian Inference is known to be a hard problem, for Bayesian networks it is even NP-hard (Dagum and Luby, 1993). Before we explain common techniques to deal with this, let us first provide an illustrative example of Bayesian inference in practice.

One of the major applications of Bayesian Inference are *Bayesian neural networks* (Neal, 2012; Blundell et al., 2015; Gal and Ghahramani, 2016). In these networks, we assume that the weights are random variables with a prior distribution which we specify. After having seen the data, our aim is to find the posterior distribution of these weights. This allows us to quantify the uncertainty in terms of the predictions of the model, which is in general troublesome in standard neural networks and therefore its crucial limitation. In the particular case of Bayesian neural networks, we notably have that the model evidence can already become intractable for relatively simple architectures, which makes it harder to calculate expressions such as (2.1.2).

To remedy these problems, one may resort to approximate Bayesian Inference, i.e. try to build a good approximation of the posterior distribution. This approach generally falls into two broad categories: *(i)* Monte Carlo methods (such as MCMC (Neal, 1993), Importance Sampling (Kloek and Van Dijk, 1978), and Sequential Monte Carlo (Doucet et al., 2001)) which are based on *sampling* and *(ii)* Variational Inference methods (such as Variational Bayes (Jordan et al., 1998) and Expectation Propagation (Minka, 2013)) which are rooted in *optimisation* techniques.

Major advantages of Variational Inference include that it tends to be faster and scales easier to high-dimensional data (Blei and Jordan, 2006; Blei et al., 2017). The **V**ariational **A**uto **E**ncoders (VAEs) by Kingma and Welling (2013) that exploit Variational Inference are a good example of this. VAEs are deep generative models which are particularly effective for large, high–dimensional datasets. However, an issue with Variational Inference is that it relies on optimisation methods which might not be able to reach the true distribution. Therefore, there exist few theoretical guarantees. On the contrary, for Monte Carlo methods, there is a rich literature of results that provide theoretical justification (Robert and Casella, 1999). As a result, a great deal of current research is focused on establishing theoretical guarantees for scalable Variational Inference (Alquier et al., 2016; Alquier and Ridgway, 2020), which is also part of the work done in this dissertation.

Before we delve into these theoretical results, let us start by outlining the methodology for performing Variational Inference.

2.2 Variational Inference methods for Bayesian Inference

2.2.1 Traditional Variational Inference and its limitations

In Variational Inference, we regard the approximation problem as an optimisation problem. For this problem, we introduce a variational family $\mathcal{Q}$ of probability densities and find the probability density that is "closest" to the posterior density. In traditional Variational Inference, the "closeness" of two distributions is measured using the **K**ullback-**L**eibler (KL) divergence (Kullback and Leibler, 1951), whose definition will be given now.

Definition 1. *Let $\mathbb{Q}$ and $\mathbb{P}$ be probability measures on $(\mathsf{Y}, \mathcal{Y})$ that are absolutely continuous with respect to ν, that is, $\mathbb{Q} \preceq \nu$ and $\mathbb{P} \preceq \upsilon$. Moreover, we denote by $q = \frac{\mathrm{d}\mathbb{Q}}{\mathrm{d}\nu}$ and $p = \frac{\mathrm{d}\mathbb{P}}{\mathrm{d}\nu}$ the Radon-Nikodym derivatives of $\mathbb{Q}$ and $\mathbb{P}$ with respect to ν. Then, the KL divergence between $\mathbb{Q}$ and $\mathbb{P}$ is defined as*

$$\mathbb{D}_{\mathrm{KL}}(\mathbb{Q} \parallel \mathbb{P}) = \mathbb{E}_q\left[\log\left(\frac{q(Y)}{p(Y)}\right)\right] = \int_{\mathsf{Y}} \log\left(\frac{q(y)}{p(y)}\right) q(y)\,\nu(\mathrm{d}y).$$

We remark that in this dissertation, only the Lebesgue measure will be utilised, in which case the Radom-Nikodym derivatives q and p are simply the corresponding probability densities to $\mathbb{Q}$ and $\mathbb{P}$.

In traditional Variational Inference, we now seek to find the variational density $q^*(y)$, i.e. the density that is closest in KL divergence to the posterior density. We intent to do this by solving the following optimisation problem

$$\inf_{q\in\mathcal{Q}} \mathbb{D}_{\mathrm{KL}}(\mathbb{Q} \parallel \mathbb{P}_{|\mathcal{D}}) = \inf_{q\in\mathcal{Q}} \int_{\mathsf{Y}} \log\left(\frac{q(y)}{p(y|\mathcal{D})}\right) q(y)\,\nu(\mathrm{d}y), \tag{2.2.1}$$

where $q(\cdot)$ and $p(\cdot|\mathcal{D})$ are the respective probability densities to $\mathbb{Q}$ and $\mathbb{P}_{|\mathcal{D}}$. Some important properties of the KL divergence are summarised in the following proposition.

Proposition 1 (Kullback and Leibler (1951))**.**

- $\mathbb{D}_{\mathrm{KL}}(\mathbb{Q} \parallel \mathbb{P}) \geqslant 0$ *(Gibbs' inequality),*
- $\mathbb{D}_{\mathrm{KL}}(\mathbb{Q} \parallel \mathbb{P}) = 0 \iff \mathbb{Q} = \mathbb{P}$,
- *In general,* $\mathbb{D}_{\mathrm{KL}}(\mathbb{Q} \parallel \mathbb{P}) \neq \mathbb{D}_{\mathrm{KL}}(\mathbb{P} \parallel \mathbb{Q})$.

Notice that Proposition 1 reveals the motivation to use the KL divergence as a dissimilarity measure. It is only zero when the probability measures coincide and will otherwise be strictly positive. When the distributions differ significantly, we also anticipate that the expected logarithmic difference between $q(\cdot)$ and $p(\cdot|\mathcal{D})$ in (2.2.1) is larger.

A possible issue with (2.2.1) is that we do not know $p(y|\mathcal{D})$. Fortunately, we can solve an equivalent optimisation problem in which only the joint density $p(y, \mathcal{D})$ appears. Note that

$$\begin{aligned}\mathbb{D}_{\mathrm{KL}}(\mathbb{Q} \parallel \mathbb{P}_{|\mathcal{D}}) &= \int_{\mathsf{Y}} \log\left(\frac{q(y)}{p(y,\mathcal{D})}\right) q(y)\,\nu(\mathrm{d}y) + \log p(\mathcal{D}) \\ &= -\mathcal{L}(q;\mathcal{D}) + \log p(\mathcal{D}),\end{aligned} \tag{2.2.2}$$

where we introduced the notation

$$\mathcal{L}(q;\mathcal{D}) := \int_{\mathsf{Y}} \log\left(\frac{p(y,\mathcal{D})}{q(y)}\right) q(y)\,\nu(\mathrm{d}y). \tag{2.2.3}$$

Notice that the log model evidence $\log p(\mathcal{D})$ does not depend on q and can therefore be regarded as a constant in this optimisation problem. Hence, minimising the KL divergence as in (2.2.1) is equivalent to the following maximisation problem

$$\sup_{q\in\mathcal{Q}} \mathcal{L}(q;\mathcal{D}). \tag{2.2.4}$$

In the literature, $\mathcal{L}(q;\mathcal{D})$ is usually referred to as the **E**vidence **L**ower **BO**und (ELBO). The name follows from (2.2.2) where we observe that $\mathcal{L}(q;\mathcal{D}) \leqslant \log p(\mathcal{D})$ since $\mathbb{D}_{\mathrm{KL}}(\mathbb{Q} \parallel \mathbb{P}) \geqslant 0$. Hence, the ELBO is a lower bound on the log model evidence[1]. Therefore, many Variational Inference algorithms directly maximise the ELBO as this does not contain the unknown posterior density $p(\cdot|\mathcal{D})$.

Another remark has to be made about the asymmetry of the divergence measure, i.e., $\mathbb{D}_{\mathrm{KL}}(\mathbb{Q} \parallel \mathbb{P}) \neq \mathbb{D}_{\mathrm{KL}}(\mathbb{P} \parallel \mathbb{Q})$. The first one, $\mathbb{D}_{\mathrm{KL}}(\mathbb{Q} \parallel \mathbb{P})$, is what we also call the *exclusive* KL divergence. This one is most commonly seen in practice, as we can get rid of the normalisation constant in the optimisation problem using (2.2.3), which only contains the joint density. Subsequently, when the joint is written as a product over a lot of data, we can resort to mini-batching to obtain computationally cheap estimates (Zhang et al., 2018). Nonetheless, one could also attempt to minimise the *inclusive* KL divergence, given by $\mathbb{D}_{\mathrm{KL}}(\mathbb{P} \parallel \mathbb{Q})$. This procedure is often referred to as *expectation propagation* (Minka, 2013). It can possibly lead to better approximations but it is in general computationally more expensive and therefore less used in practice (Li et al., 2015).

Mean-field approximating family

Next to that, in Variational Inference, one has to specify a variational family $\mathcal{Q}$. The aim is to choose the variational family broad enough to closely approximate complex distributions but simultaneously keep the optimisation problem computationally feasible. A classical choice for $\mathcal{Q}$ is to work within the *Mean-field approximating family* (Parisi and Shankar, 1988). For a d-dimensional density, this means that we assume that the variational density factorises in terms of its components:

$$\mathcal{Q} = \left\{ q : y \mapsto \prod_{\ell=1}^{d} q_\ell(y_\ell) \right\}$$

where each latent variable y_ℓ is dictated by its own variational density q_ℓ. When we plug this factorised variational density into the ELBO (2.2.3) and write it as a function of q_ℓ we can derive an optimal update for that coordinate as shown in the following proposition.

Proposition 2. *Consider the maximisation problem in* (2.2.4)*. Within the Mean-field approximating family and given fixed factors $q_j(y_j)$ for $j \neq \ell$, the optimal variational density for component ℓ is given by*

$$q_\ell^*(y_\ell) \propto \exp\left(\mathbb{E}_{-\ell}[\log p(Y,\mathcal{D})]\right), \tag{2.2.5}$$

[1]Here we say "the" ELBO as it is the standard ELBO when using the KL divergence. However, it has to be noted that the ELBO is not unique and there are many different expressions used in practice.

where $Y_{-\ell}$ denotes the vector Y without the ℓth component and $\mathbb{E}_{-\ell}$ the expectation with respect to q omitting the factor q_ℓ.

The proof of Proposition 2 can be found in for instance Bishop (2006) and Blei et al. (2017) but is also included in Appendix A.1 for the sake of completeness. An algorithm based on Proposition 2, that is often employed in an attempt to maximise the ELBO is the **C**oordinate **A**scent **V**ariational **I**nference (CAVI) algorithm which is outlined in Appendix A.2.

Limitations

However, there are limitations to this Variational Inference approach. In the context of this dissertation, we will in particular be interested in two specific limitations of this approach.

(i) *A search space that is too restrictive to match the target at the end of the optimisation procedure*. A good example to illustrate this shortcoming is the Mean-field approximating family. The drawback of this family is that we can only find posterior densities which have no correlation between the components. This might lead to poor approximations. To clarify this, consider Figure 2.1 in which we attempt to approximate a strongly correlated bivariate Gaussian distribution. Here, the Mean-field approximating family is employed and we see that for both the exclusive and inclusive KL, the means coincide but the covariance structure is substantially different. As we are not modelling the correlation between the components, the variational density is also not able to capture this. Hence, the search space is too restrictive for this target density.

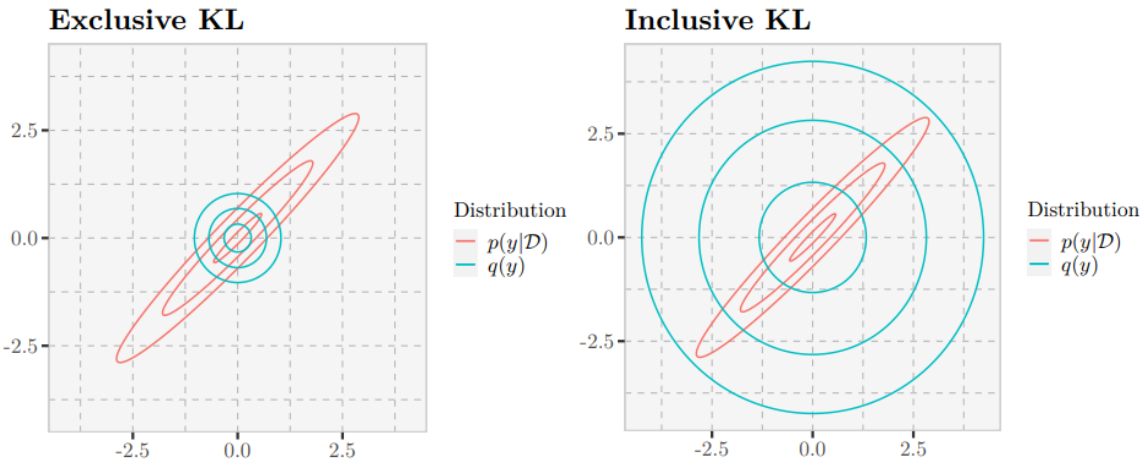


Figure 2.1: Correlated bivariate normal distribution and its approximations using exclusive KL (left) and inclusive KL (right) while using the product of two univariate Gaussians as variational density. The contours in this plot correspond to 1, 2, and 3 standard deviations of the distribution. Figure was inspired by Bishop (2006), further details can be found in Appendix D.1.3.

(ii) *An inappropriate choice of the objective function appearing in the optimisation problem.* Figure 2.1 also elucidates that using difference objective functions can have a major impact on the resulting variational density as the exclusive and inclusive KL lead to substantially different approximations. In particular, when using the exclusive KL, a common issue is that the resulting variational density often underestimates the variance of the un-

derlying distribution. This can be explained when looking at the KL divergence (2.2.1) more closely. We note that the term inside the integral, i.e.

$$\log\left(\frac{q(y)}{p(y)}\right)q(y), \tag{2.2.6}$$

will be large when $q(y)$ is large and $p(y)$ is small. As we want to minimise this integral, these areas will be penalised and the optimal variational density q^* will not put mass in areas where p barely has any mass. In that sense, the exclusive KL divergence is *mode seeking*, just as we see on the left in Figure 2.1. For the inclusive KL divergence, this is the other way around and it penalises areas where p has large mass but q does not. Thus, we sometimes say the inclusive KL is *mode covering*. These peculiarities about the exclusive and inclusive KL divergence illustrate why choosing an appropriate objective function is highly pertinent to Variational Inference in general.

Following the research of Daudel et al. (2021), this dissertation will address the first two issues by introducing a broader variational family and using a more general divergence measure. Before we delve into this paper, let us first review this more general divergence measure, that is, the α-divergence.

2.2.2 Variational Inference within the α-divergence family

The α-divergence family is a family of divergence measures which includes the exclusive and inclusive KL divergence. There is already a branch in Variational Inference that focuses on minimising this divergence measure with seminal work including Minka (2004), Minka et al. (2005) and Li and Turner (2016). The definition is given below.

Definition 2. *Let $\mathbb{Q}$ and $\mathbb{P}$ be probability measures on $(\mathsf{Y}, \mathcal{Y})$ that are absolutely continuous with respect to ν, that is, $\mathbb{Q} \preceq \nu$ and $\mathbb{P} \preceq \nu$. Moreover, we denote by $q = \frac{\mathrm{d}\mathbb{Q}}{\mathrm{d}\nu}$ and $p = \frac{\mathrm{d}\mathbb{P}}{\mathrm{d}\nu}$ the Radon-Nikodym derivatives of $\mathbb{Q}$ and $\mathbb{P}$ with respect to ν. Then, the α-divergence between $\mathbb{Q}$ and $\mathbb{P}$ is defined as*

$$\mathbb{D}_\alpha(\mathbb{Q} \,\|\, \mathbb{P}) = \int_{\mathsf{Y}} f_\alpha\left(\frac{q(y)}{p(y)}\right)p(y)\nu(\mathrm{d}y), \tag{2.2.7}$$

where

$$f_\alpha(u) = \begin{cases} \frac{1}{\alpha(\alpha-1)}[u^\alpha - 1 - \alpha(u-1)], & \text{if } \alpha \in \mathbb{R}\setminus\{0,1\}, \\ u\log(u) + 1 - u, & \text{if } \alpha = 1 \text{ (Exclusive KL)}, \\ -\log(u) + u - 1, & \text{if } \alpha = 0 \text{ (Inclusive KL)}. \end{cases} \tag{2.2.8}$$

Let us unpack this definition. Firstly, observe that the added $u-1$ in $f_\alpha(u)$ terms disappear if we restrict q to integrate to 1. The extension to the case $\alpha = 1$ comes from the fact that by L'Hôpital's rule,

$$\begin{aligned}\lim_{\alpha\to 1}\frac{1}{\alpha-1}\left[\frac{u^\alpha}{\alpha} - \frac{1}{\alpha}\right] &= \frac{\partial}{\partial\alpha}\left[\frac{u^\alpha}{\alpha} - \frac{1}{\alpha}\right]\Bigg|_{\alpha=1} \\ &= u\log(u) + 1 - u.\end{aligned}$$

In the same vein, we obtain that the limit towards $\alpha = 0$ is given by

$$\lim_{\alpha \to 0} \frac{1}{\alpha} \left[\frac{u^\alpha}{\alpha - 1} - \frac{1}{\alpha - 1} \right] = \frac{\partial}{\partial \alpha} \left[\frac{u^\alpha}{\alpha - 1} - \frac{1}{\alpha - 1} \right] \Bigg|_{\alpha = 0}$$
$$= -\log u.$$

Thus, when plugging in $u = q(y)/p(y)$, we observe that $\lim_{\alpha \to 1} \mathbb{D}_\alpha(\mathbb{Q} \parallel \mathbb{P}) = \mathbb{D}_{\mathrm{KL}}(\mathbb{Q} \parallel \mathbb{P})$ and $\lim_{\alpha \to 0} \mathbb{D}_\alpha(\mathbb{Q} \parallel \mathbb{P}) = \mathbb{D}_{\mathrm{KL}}(\mathbb{P} \parallel \mathbb{Q})$. We still define f_α for $\alpha = 1$ and $\alpha = 0$ as the function would otherwise not be defined. Since the exclusive and inclusive KL divergences are special cases of it, the α-divergence offers us more flexibility than the traditional Variational Inference approach, which motivates its use. The α-divergence is also a generalisation of several other divergences, summarised in Appendix A.3.

Let us now provide some further intuition for the parameter α. This parameter determines the *mode-seeking* behaviour of the divergence as is visualised in Figure 2.2.

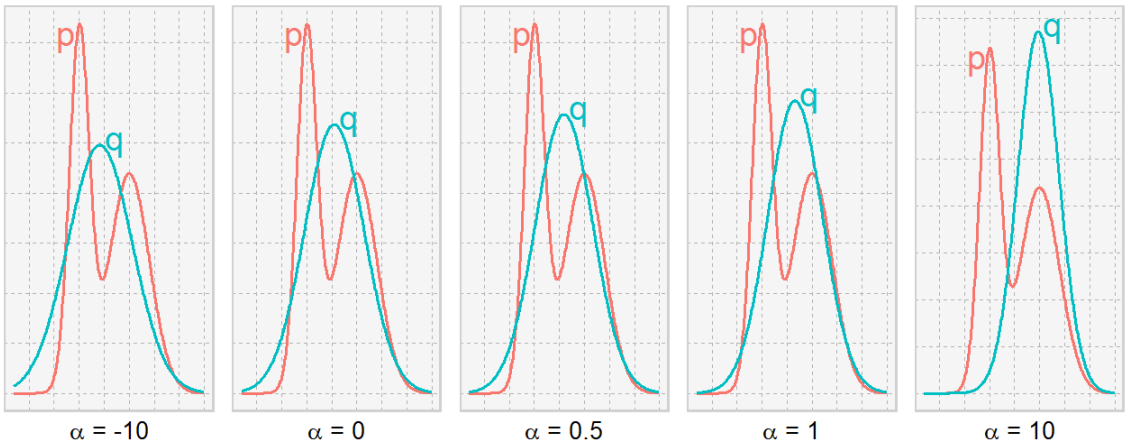


Figure 2.2: Mixture of two Gaussians p and optimal Gaussian q that minimises the α-divergence using different values of α. Figure was inspired by Minka et al. (2005), further details can be found in Appendix D.1.4. Moreover, an alternative figure that uses an unnormalised variational density is provided in Appendix A.4.

It can be observed that the larger α is, the more mode-seeking the variational density is. This means q will be attracted to the mode with the highest probability mass. For smaller α, we note that the α-divergence becomes more mode-covering. The case where $\alpha \in (0, 1)$, which will be mentioned multiple times throughout this dissertation, is a mix of both worlds. The intuition for why this behaviour occurs is the same as with the KL divergence (2.2.6).

Before moving on, let us briefly call two important results about the α-divergence.

Proposition 3 (Daudel et al. (2021))**.** *The α-divergence is always non-negative and equal to zero if and only if $\mathbb{Q}$ equals $\mathbb{P}$. Moreover, it is jointly convex in $\mathbb{Q}$ and $\mathbb{P}$ and $\forall \alpha \in \mathbb{R}, \mathbb{D}_\alpha(\mathbb{Q} \parallel \mathbb{P}) = \mathbb{D}_{1-\alpha}(\mathbb{P} \parallel \mathbb{Q})$.*

Proposition 4 (Daudel and Douc (2021))**.** *The result from (2.2.2) for the KL divergence*

also generalises to the α-divergence. That is, solving the minimisation problem

$$\inf_{q\in\mathcal{Q}} \mathbb{D}_\alpha(\mathbb{Q} \parallel \mathbb{P}) = \inf_{q\in\mathcal{Q}} \int_{\mathsf{Y}} f_\alpha\left(\frac{q(y)}{p(y|\mathcal{D})}\right) p(y|\mathcal{D})\,\nu(\mathrm{d}y)$$

is equivalent to solving

$$\inf_{q\in\mathcal{Q}} \int_{\mathsf{Y}} f_\alpha\left(\frac{q(y)}{p(y,\mathcal{D})}\right) p(y,\mathcal{D})\,\nu(\mathrm{d}y),$$

which shows we can get rid of the normalisation constant $p(\mathcal{D})$.

We refer to Appendix A.1 of Daudel and Douc (2021) for the proof of Proposition 4. Lastly, we note that α-divergence is in some sense not unique, as there exist multiple formulations in the literature. For the purpose of this dissertation, the most important alternative formulation is the so-called Rényi's α-divergence (Rényi et al., 1961; Van Erven and Harremos, 2014)

$$\begin{aligned}\mathbb{D}_\alpha^{(\mathrm{AR})}(\mathbb{Q}\|\mathbb{P}) &= \frac{1}{\alpha-1}\log\left(\int_{\mathsf{Y}} q(y)^\alpha p(y)^{\alpha-1}\nu(\mathrm{d}y)\right) \\ &= \frac{1}{\alpha-1}\log\left(1+\alpha(\alpha-1)\mathbb{D}_\alpha(\mathbb{Q}\|\mathbb{P})\right),\end{aligned} \tag{2.2.9}$$

where we note that this is a function of $\mathbb{D}_\alpha$ as defined in this dissertation. The Rényi divergence and corresponding optimisation algorithms will also be linked to the framework by Daudel et al. (2022) later.

We conclude that the α-divergence measure is also used in several state-of-the-art Variational Inference algorithms such as Hernandez-Lobato et al. (2016), Li and Turner (2016) and Kuleshov and Ermon (2017). Therefore, it provides a fruitful objective for our framework. For a more extensive discussion on the α-divergence, we refer the reader to Rényi et al., 1961; Cichocki and Amari, 2010; Van Erven and Harremos, 2014. Now that we have reviewed all essential Variational Inference techniques, we are ready to discuss the work by Daudel et al. (2022).

2.3 Review of Daudel et al. (2022)

The paper by Daudel et al. (2022) offers a novel family of iterative algorithms that can be used for Variational Inference. It tries to improve on existing algorithms by tackling the limitations outlined in Section 2.2.1. In terms of the first limitation, the α-divergence is employed, thereby offering a more flexible framework than classical KL minimisation. Moreover, in terms of the second limitation, the variational density is chosen to be a mixture distribution. In doing so, complicated multimodal densities are included in the variational family $\mathcal{Q}$. The aim is to derive an algorithm that performs α-divergence minimisation within this broad family $\mathcal{Q}$ while also being scalable.

Before we give the algorithm and the accompanying results in the next section, we formally define the optimisation problem. Let $(\mathsf{Y}, \mathcal{Y}, \nu)$ be a measured space, where ν is a σ-finite measure on the measurable space $(\mathsf{Y}, \mathcal{Y})$. Next to that, $\mathbb{P}$ denotes the probability

measure on $(\mathsf{Y}, \mathcal{Y})$ with density $p(\cdot|\mathcal{D})$ with respect to ν. Moreover, for the variational family, $\mathbb{Q}$ denotes the probability measure on $(\mathsf{Y}, \mathcal{Y})$ with density $q \in \mathcal{Q}$ with respect to ν. Now, the aim is to solve the following minimisation problem

$$\inf_{q \in \mathcal{Q}} \mathbb{D}_\alpha(\mathbb{Q} \parallel \mathbb{P}). \tag{2.3.1}$$

To do this, q is assumed to be a mixture of parametric densities. For this parametric family, $(\mathsf{T}, \mathcal{T})$ is defined to be a measurable space and $K : (\theta, A) \mapsto \int_A k(\theta, y)\nu(\mathrm{d}y)$ is a Markov transition kernel on $\mathsf{T} \times \mathcal{Y}$ with kernel density k defined on $\mathsf{T} \times \mathsf{Y}$. Hence, q is a mixture of densities of the form $y \mapsto k(\theta, y)$ where $\theta \in \mathsf{T}$.

Moreover, let $J \in \mathbb{N}$ be the number of mixture components and let $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_J)$ denote the mixture weights. Then, the mixture weights are defined on the simplex of $\mathbb{R}^J$ denoted by $\mathcal{S}_J$[2]. Furthermore, let $\Theta = (\theta_1, \ldots, \theta_J) \in \mathsf{T}^J$ be the set of all parameters. Now, using these sets, we can define the mixture model variational family

$$\mathcal{Q} = \left\{ q : y \mapsto \mu_{\boldsymbol{\lambda},\Theta} k(y) = \sum_{j=1}^{J} \lambda_j k(\theta_j, y) \ : \ \boldsymbol{\lambda} \in \mathcal{S}_J, \Theta \in \mathsf{T}^J \right\}.$$

As we have defined the divergence measure and the variational family, we can now give the optimisation problem. Using Proposition 4 we note that (2.3.1) is equivalent to the following optimisation problem

$$\begin{aligned} \inf_{q \in \mathcal{Q}} \int_{\mathsf{Y}} f_\alpha \left(\frac{q(y)}{p(y, \mathcal{D})} \right) p(y, \mathcal{D}) \nu(\mathrm{d}y) &= \inf_{\lambda \in \mathcal{S}_J, \Theta \in \mathsf{T}^J} \int_{\mathsf{Y}} f_\alpha \left(\frac{\mu_{\lambda,\Theta} k(y)}{p(y, \mathcal{D})} \right) p(y, \mathcal{D}) \nu(\mathrm{d}y) \\ &= \inf_{\boldsymbol{\lambda} \in \mathcal{S}_J, \Theta \in \mathsf{T}^J} \Psi_\alpha(\mu_{\boldsymbol{\lambda},\Theta} k(y)), \end{aligned} \tag{2.3.2}$$

where we introduced the following notation

$$\Psi_\alpha(q) = \int_{\mathsf{Y}} f_\alpha \left(\frac{q(y)}{p(y, \mathcal{D})} \right) p(y, \mathcal{D}) \nu(\mathrm{d}y).$$

Now, (2.3.2) is the main optimisation problem we will be working with. The remainder of the chapter focuses on algorithms solving this problem and corresponding results.

2.3.1 Theoretical results for a monotonic decrease under $J = 1$

To find an algorithm that solves (2.3.2), we first consider a simplified problem. That is, we do not consider a mixture model and aim at optimising the parameters of a single variational density. This is equivalent to setting $J = 1$ in (2.3.2). Hence, we focus on the problem

$$\inf_{\theta \in \mathsf{T}} \Psi_\alpha(k(\theta, \cdot)). \tag{2.3.3}$$

The paper then proposes a family of iterative algorithms that lead to a systematic decrease in Ψ_α at each step. That is, their algorithm constructs a sequence $(\theta_n)_{n \geqslant 1}$ valued in

[2]That is, all $\boldsymbol{\lambda} \in \mathbb{R}^J$ such that $\forall j \in \{1, \ldots, J\}$, $\lambda_j \geqslant 0$ and $\sum_{j=1}^J \lambda_j = 1$.

T that satisfies $\Psi_\alpha(k(\theta_{n+1},\cdot)) \leqslant \Psi_\alpha(k(\theta_n,\cdot))$ for all $n \geqslant 1$. In the paper, two procedures are introduced that satisfy this monotonic decrease: the *maximisation approach* and the *gradient-based approach*. We will now closely inspect both approaches. Before that, we give the required regularity conditions. These are:

(A1) The kernel density k on $\mathsf{T} \times \mathsf{Y}$, the function p on Y and the σ-finite measure v on $(\mathsf{Y}, \mathcal{Y})$ satisfy, for all $(\theta, y) \in \mathsf{T} \times Y$, $k(\theta, y) > 0$, $p(y) \geqslant 0$ and $0 < \int_{\mathsf{Y}} p(y)\nu(\mathrm{d}y) < \infty$.

Note that (A1) only imposes mild conditions to make sure that all functions are well-behaved. We now move on to the first approach, called the maximisation approach.

(i) Maximisation approach

We introduce the following function that is crucial in the framework

$$\varphi_n^{(\alpha)}(y) = k(\theta_n, y)^\alpha p(y)^{1-\alpha}$$
$$\check{\varphi}_n^{(\alpha)} = \frac{\varphi_n^{(\alpha)}}{\int \varphi_n^{(\alpha)} \mathrm{d}\nu}.$$

Notice that this function $\varphi_n^{(\alpha)}$ depends on n and will be updated every iteration given the value of θ_n. In the paper, a condition is derived that involves $\varphi_n^{(\alpha)}$ and guarantees a decrease in the objective function for each step. In particular, starting from some initial $\theta_1 \in \mathsf{T}$, the sequence $(\theta_n)_{n\geqslant 1}$ that satisfies

$$\int_{\mathsf{Y}} \frac{\varphi_n^{(\alpha)}(y)}{\alpha - 1} \log\left(\frac{k(\theta_{n+1}, y)}{k(\theta_n, y)}\right) \nu(\mathrm{d}y) \leqslant 0 \tag{2.3.4}$$

for all $n \geqslant 1$, yields a systematic decrease in the objective. Now, the maximisation approach is motivated by this condition and is an iterative updating scheme that satisfies this condition for each $n \geqslant 1$. This approach is summarised in the following theorem.

Theorem 1 (Daudel et al. (2022)). *(Maximisation approach) Assume (A1). Let $\alpha \in [0, 1)$ and let $(b_n)_{n\geqslant 1}$ be a non-negative sequence. Starting from an initial $\theta_1 \in \mathsf{T}$, let $(\theta_n)_{n\geqslant 1}$ be defined iteratively as follows*

$$\theta_{n+1} = \underset{\theta\in\mathsf{T}}{\operatorname{argmax}} \int_{\mathsf{Y}} \left[\varphi_n^{(\alpha)}(y) + b_n k\left(\theta_n, y\right)\right] \log\left(\frac{k(\theta, y)}{k\left(\theta_n, y\right)}\right) \nu(\mathrm{d}y), \quad n \geqslant 1, \tag{2.3.5}$$

where we assume that this argmax is uniquely defined at each step. Then condition (2.3.4) holds and we obtain a systematic decrease in Ψ_α at each step.

The proof can be found in the original paper but is also included in Appendix A.5. For specific choices of the kernel, simple update rules for $(\theta_n)_{n\geqslant 1}$ can now be derived from Theorem 1. Here, the sequence $(b_n)_{n\geqslant 1}$ acts as a regularisation parameter which can be chosen by the practitioner. We will now introduce a Gaussian example and demonstrate the updates following from the maximisation approach. This example will later come back in our own theoretical contributions in Chapter 3.

Example 1 (Maximisation approach for a Gaussian density). *Let $\mathsf{Y} = \mathbb{R}^d$ and ν be the Lebesgue measure. Moreover, we let the kernel be Gaussian, that is, we set $k(\theta, y) =$*

$\mathcal{N}(y; m, \Sigma)$ where $\theta = (m, \Sigma) \in \mathsf{T}$ denote the mean and covariance matrix. Besides, T includes $\mathbb{R}^d$ and all positive definite symmetric matrices in $\mathbb{R}^{d\times d}$. Assume that

$$0 < \int_{\mathsf{Y}} \left(1 + \|y\|^{2/(1-\alpha)}\right) p(y)\,\mathrm{d}y < \infty, \tag{2.3.6}$$

where $\|\cdot\|$ denotes the Euclidean norm. Then (A1) holds. For this example, it has been derived that the argmax problem from Theorem 1 has the following unique solution

$$\begin{aligned} m_{n+1} &= \gamma_n \widehat{m}_n + (1-\gamma_n)\, m_n \\ \Sigma_{n+1} &= \gamma_n \widehat{\Sigma}_n + (1-\gamma_n)\,\Sigma_n + \gamma_n\,(1-\gamma_n)\,(\widehat{m}_n - m_n)\,(\widehat{m}_n - m_n)^T \end{aligned} \tag{2.3.7}$$

where all terms are defined by

$$\begin{aligned} \gamma_n &= \frac{1}{1+b_n} \\ \widehat{m}_n &= \int_{\mathsf{Y}} y\; \breve{\varphi}_n^{(\alpha)}(y)\,\mathrm{d}y \\ \widehat{\Sigma}_n &= \int_{\mathsf{Y}} yy^T\; \breve{\varphi}_n^{(\alpha)}(y)\,\mathrm{d}y - \widehat{m}_n \widehat{m}_n^T. \end{aligned} \tag{2.3.8}$$

Let us now give some interpretation of this updating scheme. First of all, we see that b_n solely determines $\gamma_n \in (0, 1]$ where a smaller value of γ_n represents more regularisation. Moreover, $\widehat{m}_n$ and $\widehat{\Sigma}_n$ can be interpreted as the "proposed" new mean and covariance matrix, respectively. Here, γ_n determines the degree in which we move towards these new parameters. The detailed derivation of (2.3.7) is not provided in Daudel et al. (2022) and can therefore be found in Appendix A.6.

(ii) Gradient-based approach

We now present an alternative updating scheme proposed by the paper that also leads to a systematic decrease in $\Psi(k(\theta, \cdot))$. This algorithm adopts several ideas from the Gradient Descent algorithm which we first briefly recall.

The Gradient Descent Algorithm is an iterative algorithm that traces back to Cauchy et al. (1847) and is used to find minima of a differentiable objective function g. Starting from some initial value θ_1, it uses the following updates

$$\theta_{n+1} = \theta_n - r_n \nabla g(\theta_n), \tag{2.3.9}$$

where r_n is the learning rate, which does not need to be constant. The idea is that each iteration we take a step in the direction of steepest descent and by doing that we aim to find the minimum after several iterations. Applying this to the α-divergence and Rényi's α-divergence (2.2.9) leads respectively to the following updates

$$\begin{aligned} \theta_{n+1} &= \theta_n - r_n \int_{\mathsf{Y}} \frac{\varphi_n^{(\alpha)}(y)}{\alpha - 1} \frac{\partial \log k(\theta, y)}{\partial \theta}\bigg|_{(\theta,y)=(\theta_n,y)} \nu(\mathrm{d}y), \quad n \geqslant 1 \\ \theta_{n+1} &= \theta_n - r_n \int_{\mathsf{Y}} \frac{\breve{\varphi}_n^{(\alpha)}(y)}{\alpha - 1} \frac{\partial \log k(\theta, y)}{\partial \theta}\bigg|_{(\theta,y)=(\theta_n,y)} \nu(\mathrm{d}y), \quad n \geqslant 1, \end{aligned} \tag{2.3.10}$$

where we refer to Appendix A.2.2 of Daudel et al. (2022) for the derivation.

We now introduce the alternative updating scheme that is based on the idea of Gradient Descent proposed by Daudel et al. (2022). In this approach, a sequence of auxiliary functions $(g_n)_{n\geqslant 1}$ is introduced that bear similarities with the objective function. For each n, we then take a gradient step based on g_n. Under additional smoothness conditions (see Appendix B.4 for the definition and further illustration of the concept of β-smoothness), the paper also demonstrates that this approach leads to a monotonic decrease in Ψ_α. This is summarised in the following theorem

Theorem 2 (Daudel et al. (2022))**.** *(Gradient-based approach) Assume (A1). Let $\mathsf{T} \subseteq \mathbb{R}^d$ be a convex set, let $\alpha \in [0, 1)$ and let $(\gamma_n)_{n\geqslant 1}$ be valued in $(0, 1]$. Starting from an initial $\theta_1 \in \mathsf{T}$, let $(\theta_n)_{n\geqslant 1}$ be defined iteratively as follows*

$$\theta_{n+1} = \theta_n - \frac{\gamma_n}{\beta_n} \nabla g_n(\theta_n), \quad n \geqslant 1,$$

where $(g_n)_{n\geqslant 1}$ is the sequence of functions defined by: for all $n \geqslant 1$ and all $\theta \in \mathsf{T}$,

$$g_n(\theta) = \int_{\mathsf{Y}} \frac{\varphi_n^{(\alpha)}(y)}{\alpha - 1} \log\left(\frac{k(\theta, y)}{k(\theta_n, y)}\right) \nu(\mathrm{d}y) \tag{2.3.11}$$

and g_n is assumed to be β_n-smooth. Then, Theorem 1 applies and we get a systematic decrease in Ψ_α.

The proof follows trivially from Lemma 4 in the appendix but can also be found in the original paper. To apply this method, we need the derivative of g_n. We can easily derive that for all $n \geqslant 1$ and all $\theta' \in \mathsf{T}$,

$$\nabla g_n(\theta') = \int_{\mathsf{Y}} \frac{\varphi_n^{(\alpha)}(y)}{\alpha - 1} \frac{\partial \log k(\theta, y)}{\partial \theta}\bigg|_{(\theta,y)=(\theta',y)} \nu(\mathrm{d}y).$$

Hence the update from Theorem 2 now becomes

$$\theta_{n+1} = \theta_n - \frac{\gamma_n}{\beta_n} \int_{\mathsf{Y}} \frac{\varphi_n^{(\alpha)}(y)}{\alpha - 1} \frac{\partial \log k(\theta, y)}{\partial \theta}\bigg|_{(\theta,y)=(\theta_n,y)} \nu(\mathrm{d}y), \quad n \geqslant 1,$$

where we notice that a similarity with the updates in (2.3.10). The crucial difference with Gradient Descent is that the gradient-based approach takes steps based on a function g_n that updates every iteration whereas Gradient Descent uses a constant objective g. Although, we are applying gradient steps to different functions, we do get similar derivatives. We now exhibit this connection between Gradient Descent and the gradient-based approach for a Gaussian variational density.

Example 2 (Gradient-based approach for a Gaussian density)**.** *Let us adopt the assumptions from Example 1. Moreover, we further simplify this example by looking at kernel $k(\theta, y) = \mathcal{N}(y; m, \Sigma)$ where the covariance matrix is now known and fixed $\Sigma = \sigma^2 I_d$ with $\sigma^2 > 0$. Hence, we only optimise parameter $\theta = m$. As derived in Appendix A.7 and*

in the original paper, this yields the following update:

$$\begin{aligned} m_{n+1} &= m_n - \frac{\gamma_n}{\beta_n}\frac{\int \varphi_n^{(\alpha)}\,\mathrm{d}\nu}{(1-\alpha)}\Sigma^{-1}\left(m_n - \int_{\mathsf{Y}} y\check{\varphi}(y)\mathrm{d}y\right) \\ &= \left(1 - \gamma_n \frac{\int \varphi_n^{(\alpha)}\,\mathrm{d}\nu}{\beta_n\sigma^2(1-\alpha)}\right) m_n + \gamma_n \frac{\int \varphi_n^{(\alpha)}\,\mathrm{d}\nu}{\beta_n\sigma^2(1-\alpha)}\widehat{m}_n. \end{aligned} \tag{2.3.12}$$

Now as also shown in Appendix A.7, in this Gaussian case g_n is β_n-smooth where $\beta_n = \sigma^{-2}(1-\alpha)^{-1}\int \varphi_n^{(\alpha)}\,\mathrm{d}\nu$. Therefore, the above update for the mean simplifies to

$$m_{n+1} = \gamma_n \widehat{m}_n + (1-\gamma_n)m_n, \tag{2.3.13}$$

which we also saw in (2.3.7). Note that the gradient-based approach requires stronger assumptions in terms of the β_n-smoothness than the maximisation approach but it leads to the same updating scheme in the Gaussian case. As shown in Daudel et al. (2022), it is harder to derive an updating scheme for Σ as it requires a non-constant smoothing index β_n and therefore we also refrain from this in this dissertation. Moreover, this draws a connection with the Gradient Descent updates in (2.3.10). This is because the update (2.3.13) can be interpreted as Gradient Descent step for Rényi's α-divergence minimisation with a learning rate $r_n = \sigma^2(1-\alpha)\gamma_n$, where $\gamma_n \in [0,1)$. This connection will prove to be useful later in the theoretical contributions in Chapter 3 where link the derived convergence results with convergence results from the Gradient Descent literature. Before delving into this, let us present the algorithm from Daudel et al. (2022) to carry out α-divergence minimisation for Gaussian Mixture Models (GMMs).

2.3.2 Extension to Gaussian Mixture Models (GMMs)

Let us now consider the full optimisation problem from equation (2.3.2), that is, we will be using a mixture distribution as the variational distribution. The goal is now to derive iterative schemes that optimise both the mixture weights and mixture component parameters $(\boldsymbol{\lambda}_n, \Theta_n)_{n\geqslant 1}$. The difficulty here is the optimisation over the mixture weights $\boldsymbol{\lambda}$, which lie in the constrained space $\mathcal{S}_J$ and which we want to optimise simultaneously with Θ. Yet, Daudel et al. (2022) enables that by deriving independent conditions on $\boldsymbol{\lambda}_{n+1}$ and Θ_{n+1} given the values of $(\boldsymbol{\lambda}_n, \Theta_n)$ at iteration n that lead to a systematic decrease in Ψ_α. We omit the proofs in this section for the sake of conciseness and refer the interested reader to Daudel et al. (2022).

We first introduce the following shorthand notation for the variational density $\mu_n k = \mu_{\boldsymbol{\lambda}_n,\Theta_n} k$. Next to that, we denote

$$\begin{aligned} \varphi_{j,n}^{(\alpha)}(y) &= k(\theta_{j,n}, y)\left(\frac{\mu_n k(y)}{p(y)}\right)^{\alpha-1} \\ \check{\varphi}_{j,n}^{(\alpha)} &= \frac{\varphi_{j,n}^{(\alpha)}}{\int \varphi_{j,n}^{(\alpha)}\,\mathrm{d}\nu}. \end{aligned}$$

Note that $\varphi_{j,n}^{(\alpha)}(y)$ now also depends on mixture component j. For $J = 1$, it simplifies to $\varphi_n^{(\alpha)}(y)$ from the previous section.

For the mixture case, a similar approach is taken as in the previous section. Again, a condition similar to (2.3.4) is derived that ensures a monotonic decrease in Ψ_α. We first demonstrate the key result for the mixture weight optimisation, given in the following theorem.

Theorem 3 (Daudel et al. (2022))**.** *Assume (A1). Let $\alpha \in [0, 1)$, let $(\eta_n)_{n\geqslant 1}$ be valued in $(0, 1]$ and let $(\kappa_n)_{n\geqslant 1}$ be such that $(\alpha-1)\kappa_n \geqslant 0$ at all times n. Furthermore, let $(\Theta_n)_{n\geqslant 1}$ be any sequence valued in T^J. Starting from an initial $\boldsymbol{\lambda}_1 \in \mathcal{S}_J^+$, let $(\boldsymbol{\lambda}_n)_{n\geqslant 1}$ be defined iteratively such that for all $n \geqslant 1$*

$$\lambda_{j,n+1} = \frac{\lambda_{j,n}\left[\int_Y \varphi_{j,n}^{(\alpha)}(y)\nu(\mathrm{d}y) + (\alpha-1)\kappa_n\right]^{\eta_n}}{\sum_{\ell=1}^J \lambda_{\ell,n}\left[\int_Y \varphi_{\ell,n}^{(\alpha)}(y)\nu(\mathrm{d}y) + (\alpha-1)\kappa_n\right]^{\eta_n}}, \quad j = 1 \dots J.$$

Then, $\Psi_\alpha(\mu_{n+1}k) \leqslant \Psi_\alpha(\mu_n k)$ for all $n \geqslant 1$.

Now, in terms of the optimisation of the mixture component parameters Θ, both the maximisation and gradient-based approach can be generalised to the mixture case while maintaining the systematic decrease in Ψ_α property. For the detailed derivations, we refer to the original paper and Appendix A.8. In the particular case of GMMs, let us now present the updates that are obtained.

For this GMM, let the mixture component densities be given by d-dimensional Gaussian mixture densities $k(\theta_j, y) = \mathcal{N}(y; m_j, \Sigma_j)$, where $\theta_j = (m_j, \Sigma_j) \in \mathsf{T}$ denotes the mean and covariance matrix of the j-th Gaussian component. Moreover, we assume the target p satisfies (2.3.6). Then, a valid update for the mixture weights follows from Theorem 3. For the mixture component parameter updates, let us introduce the following two quantities

$$\widehat{m}_{j,n} = \int_{\mathsf{Y}} y\, \breve{\varphi}_{j,n}^{(\alpha)}(y)\nu(\mathrm{d}y) \quad \text{and} \quad \widehat{\Sigma}_{j,n} = \int_Y yy^T\, \breve{\varphi}_{j,n}^{(\alpha)}(y)\nu(\mathrm{d}y) - \widehat{m}_{j,n}\widehat{m}_{j,n}^T.$$

The generalised maximisation approach and gradient-based approach now give the following updates for the mean for all $j = 1, \dots, J$ and $n \geqslant 1$,

$$m_{j,n+1} = \gamma_{j,n}\widehat{m}_{j,n} + (1-\gamma_{j,n})\, m_{j,n}, \tag{2.3.14}$$

which mimics the mean updates in (2.3.7) and (2.3.13) where we now allow $\gamma_{j,n}$ to depend on j. Next to that, the maximisation approach enables covariance updates like previously

$$\Sigma_{j,n+1} = \gamma_{j,n}\widehat{\Sigma}_{j,n} + (1-\gamma_{j,n})\,\Sigma_{j,n} + \gamma_{j,n}\,(1-\gamma_{j,n})\,(\widehat{m}_{j,n} - m_{j,n})\,(\widehat{m}_{j,n} - m_{j,n})^T.$$

This time, the link between mean updates and Gradient Descent schemes can be drawn by selecting a specific learning rate, which leads to the **R**ényi **G**radient **D**escent (RGD) update

$$m_{j,n+1} = m_{j,n} + \gamma_{j,n}\frac{\int_{\mathsf{Y}} \lambda_{j,n}\varphi_{j,n}^{(\alpha)}(y)\,(y - m_{j,n})\,\nu(\mathrm{d}y)}{\int_{\mathsf{Y}} \mu_n\, k(y)^\alpha p(y)^{1-\alpha}\nu(\mathrm{d}y)}, \quad j = 1 \dots J, \tag{2.3.15}$$

where we refer to Example 4 in Daudel et al. (2022) for the derivation of (2.3.15). On the other hand, the mean update in (2.3.14) will be referred to as the **M**aximisation **G**radient (MG) update.

However, one should note that all integrals involved in these updates are of the form

$$\int_{\mathsf{Y}} \varphi_{j,n}^{(\alpha)}(y) g(y) \nu(\mathrm{d}y) \tag{2.3.16}$$

for which we do not have a closed form solution. Therefore, we resort to stochastic approximations. We will make use of an Importance Sampling estimator (Kloek and Van Dijk, 1978; Robert and Casella, 1999) to estimate (2.3.16). Given M samples $(Y_{m,n})_{1\leqslant m\leqslant M}$ from some proposal q_n at time n, this estimator would take the following form

$$\frac{1}{M}\sum_{m=1}^{M} \frac{\varphi_{j,n}^{(\alpha)}(Y_{m,n})}{q_n(Y_{m,n})} g(Y_{m,n}) = \frac{1}{M}\sum_{m=1}^{M} \widehat{\varphi}_{j,n}^{(\alpha)}(Y_{m,n}) g(Y_{m,n}),$$

where $\widehat{\varphi}_{j,n}(y) = \frac{\varphi_{j,n}^{(\alpha)}(y)}{q_n(y)} = \frac{k(\theta_{j,n},y)}{q_n(y)} \left(\frac{\mu_n k(y)}{p(y)}\right)^{\alpha-1}$. The paper proposes two choices for proposal q_n. That is, *(i)* the **best sampler at time** n **(IS-n)** which is given by $q_n = \mu_n k$. This means we are using the best approximation of p we have at time n. Or, *(ii)* the **Uniform sampler (IS-unif)** which is given by $q_n = J^{-1}\sum_{j=1}^{J} k(\theta_{j,n},\cdot)$. This sampler ensures fair sampling among all components.

We conclude by giving the full GMMs optimisation procedure in Algorithm 1 below. Let us remark that the framework does allow for a learning rate $\gamma_{j,n}$ that depends on component j, however Algorithm 1 only focuses on the case $\gamma_{j,n} := \gamma_n$.

Algorithm 1: GMMs optimisation with the IS-n/IS-unif sampler

At iteration n,

1. Draw independently M samples $(Y_{m,n})_{1\leqslant m\leqslant M}$ from the proposal q_n.
2. For all $j = 1 \dots J$,

(a) Compute

$$\widehat{m}_{j,n} = \frac{\sum_{m=1}^{M} \widehat{\varphi}_{j,n}^{(\alpha)}(Y_{m,n}) \cdot Y_{m,n}}{\sum_{m=1}^{M} \widehat{\varphi}_{j,n}^{(\alpha)}(Y_{m,n})} \text{ and } \widehat{\Sigma}_{j,n} = \frac{\sum_{m=1}^{M} \widehat{\varphi}_{j,n}^{(\alpha)}(Y_{m,n}) \cdot Y_{m,n} Y_{m,n}^T}{\sum_{m=1}^{M} \widehat{\varphi}_{j,n}^{(\alpha)}(Y_{m,n})} - \widehat{m}_{j,n}\widehat{m}_{j,n}^T$$

(b) Choose one between the (MG) and (RGD) approaches

$$(MG) \quad m_{j,n+1} = (1-\gamma_n)\, m_{j,n} + \gamma_n \widehat{m}_{j,n}$$

$$(RGD) \quad m_{j,n+1} = m_{j,n} + \gamma_n \frac{\lambda_{j,n} \sum_{m=1}^{M} \widehat{\varphi}_{j,n}^{(\alpha)}(Y_{m,n}) \cdot (Y_{m,n} - m_{j,n})}{\sum_{j=1}^{J}\sum_{m=1}^{M} \lambda_{j,n} \widehat{\varphi}_{j,n}^{(\alpha)}(Y_{m,n})}$$

and set

$$\lambda_{j,n+1} = \frac{\lambda_{j,n}\left[\sum_{m=1}^{M} \widehat{\varphi}_{j,n}^{(\alpha)}(Y_{m,n}) + (\alpha-1)\kappa_n\right]^{\eta_n}}{\sum_{\ell=1}^{J} \lambda_{\ell,n}\left[\sum_{m=1}^{M} \widehat{\varphi}_{\ell,n}^{(\alpha)}(Y_{m,n}) + (\alpha-1)\kappa_n\right]^{\eta_n}}$$

$$\Sigma_{j,n+1} = \gamma_n \widehat{\Sigma}_{j,n} + (1-\gamma_n)\,\Sigma_{j,n} + \gamma_n(1-\gamma_n)(\widehat{m}_{j,n} - m_{j,n})(\widehat{m}_{j,n} - m_{j,n})^T.$$

2.3.3 Replicating the experimental results

We now replicate some experiments performed by the paper, which will provide us with a useful starting point for our own simulations in Chapter 4.

Target densities. In the paper, the behaviour of the algorithm is explored under three multimodal cases. Here, we use the following notation: let $\boldsymbol{u_d}$ denote the d-dimensional vector whose coordinates are all equal to 1, $\boldsymbol{I_d}$ denotes the identity matrix, and c denotes a positive constant (we set $c = 2$). We will multiply the density with such a normalisation constant c with the idea that we are in a Bayesian setting in which we do not know the normalisation constant. The multimodal cases are given by:

(i) *Equally-weighted Gaussian Mixture Model*. The target p is a mixture density of two equally-weighted d-dimensional Gaussian distributions multiplied by the positive constant c such that

$$p(y) = c \times \left[0.5\mathcal{N}\left(y; -2\boldsymbol{u_d}, \boldsymbol{I_d}\right) + 0.5\mathcal{N}\left(y; 2\boldsymbol{u_d}, \boldsymbol{I_d}\right)\right].$$

(ii) *Imbalanced Gaussian Mixture Model*. The target p is a mixture density of three d-dimensional Gaussian distributions with unequal weights and multiplied by the positive constant c such that

$$p(y) = c \times \left[0.35\mathcal{N}\left(y; -2\boldsymbol{u_d}, \boldsymbol{I_d}\right) + 0.25\mathcal{N}\left(y; 2\boldsymbol{u_d}, \boldsymbol{I_d}\right) + 0.4\mathcal{N}\left(y; \boldsymbol{u_d}, \boldsymbol{I_d}\right)\right].$$

(iii) *Equally-weighted Student's t Mixture Model*. The target p is a mixture density of two equally-weighted d-dimensional Student's t distributions with two degrees of freedom (i.e. $a = 2$) multiplied by the positive constant c such that

$$p(y) = c \times \left[0.5\,\mathrm{t}(y; -2\boldsymbol{u_d}, \boldsymbol{I_d}, a) + 0.5\,\mathrm{t}(y; 2\boldsymbol{u_d}, \boldsymbol{I_d}, a)\right].$$

Performance criterion. To assess the convergence of the algorithm, the **V**ariational **R**ényi (VR) bound from Li and Turner (2016) is used. This bound can be regarded as an ELBO generalised to the Rényi divergence (2.2.9). For all $q \in \mathcal{Q}$ and all $\alpha \in \mathbb{R} \setminus \{1\}$, it is given by:

$$\begin{aligned}\mathcal{L}_\alpha(q, \mathcal{D}) &= \frac{1}{1-\alpha} \log \mathbb{E}_q\left[\left(\frac{p(Y, \mathcal{D})}{q(Y)}\right)^{1-\alpha}\right] \\ &= \frac{1}{1-\alpha} \log\left(\int_{\mathsf{Y}} q(y)^\alpha p(y, \mathcal{D})^{1-\alpha} \nu(\mathrm{d}y)\right). \end{aligned} \tag{2.3.17}$$

In our settings, this integral is intractable so the Monte Carlo estimator suggested by Li and Turner (2016) has been implemented. We replicate each experiment 30 times and compute a Monte Carlo estimator of this estimator. Details on how to compute this efficiently and an explicit expression of this estimator can be found in Appendix C.1. For $\alpha \in (0, 1)$, minimising Ψ_α is equivalent to maximising the VR bound so we expect a similar monotonic increase of this bound in the experiments.

Implementation details. While we do have updating equations for the covariance matrices of the mixture components in Algorithm 1, we keep them constant at $\sigma^2 \boldsymbol{I_d}$ with $\sigma^2 = 1$

for now. Furthermore, the number of samples from q_n is set to $M = 200$ and the number of iterations is $N = 100$ such that the total computational budget is $N \times M = 20000$. Next to that, we keep $\kappa_n = 0$ for $n = 1, \ldots, N$. The GMM algorithm also requires initial values for the means and we generate these as a sample from a normal distribution with mean 0 and covariance $10\boldsymbol{I_d}$. Lastly, we set the initial mixture weights equal to $[1/J, \ldots, 1/J]$, use dimension $d = 16$, and set $\alpha = 0.2$. Analogous to the original paper, this section will only focus on varying the following factors:

- the number of mixture components: $J \in \{10, 50\}$,
- the constant value of gamma: $\gamma_n = \gamma \in \{0.1, 0.5, 1.\}$,
- the constant mixture weights learning rate: $\eta_n = \eta \in \{0, 0.05, 0.1, 0.5\}$,
- the type of sampler used for q_n, that is, either IS-n or IS-unif.

Comparing RGD and MG under $\eta = 0$.

We will first consider the case where we keep the mixture weights constant, i.e., we set $\eta = 0$. Note that the IS-n and IS-unif samplers are equivalent in this case, thus we cannot distinguish these. Therefore, we only vary J and γ. The goal of this simulation is to compare the newly introduced MG approach with the RGD approach that also follows from Li and Turner (2016) under fixed mixture weights.

The results can be found in Figure 2.3. We immediately notice a very close correspondence with Figure 1 of Daudel et al. (2022) and therefore the replication has succeeded. Moreover, we observe that the MG approach outperforms the RGD approach in all cases which exposes the potential of this novel approach. Next to that, this plot gives us interesting insights into the effect of γ on the MG approach. Generally, increasing γ leads to faster convergence. However, for case (iii) and $J = 50$, we do not see this effect.

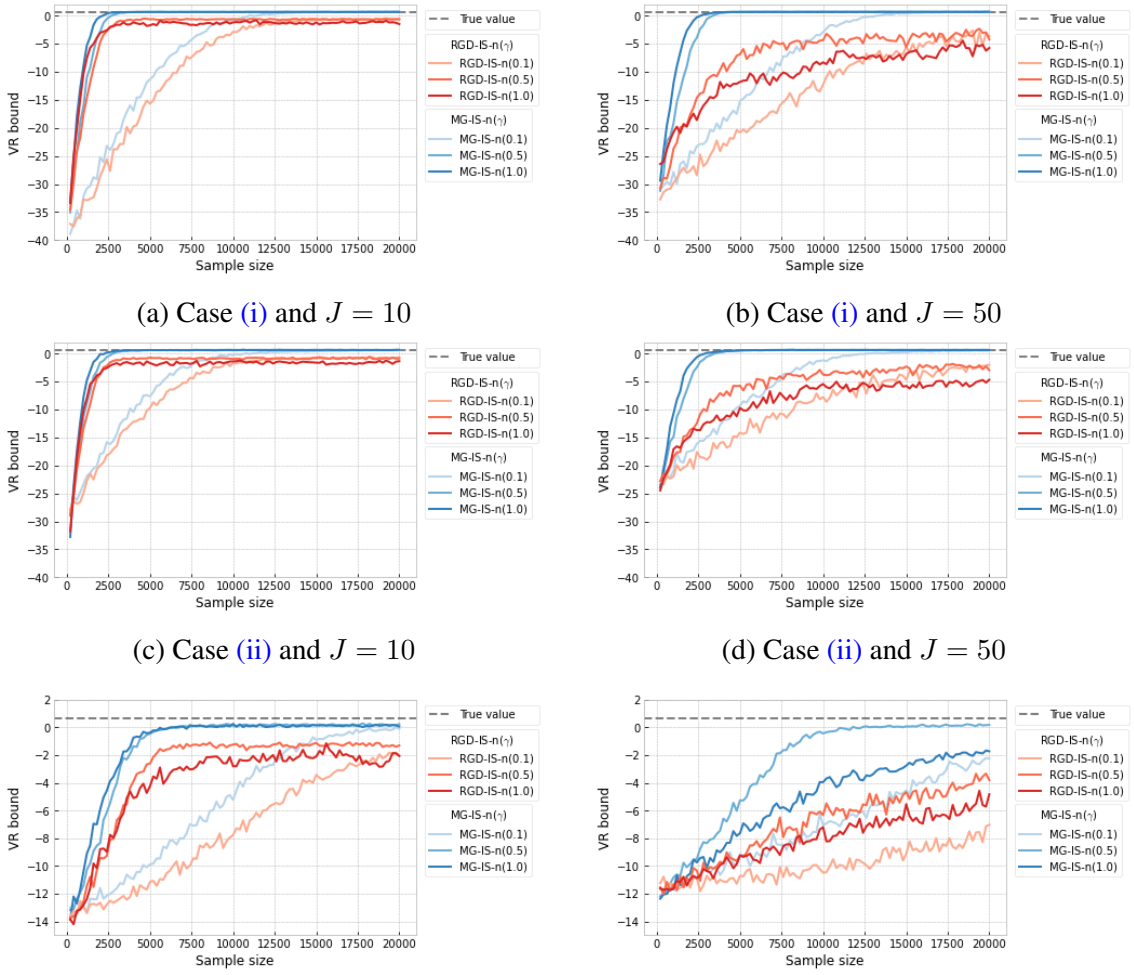
Comparing RGD and MG under $\eta > 0$.

Subsequently, the paper simulated the scenario where $\eta > 0$, i.e. the case where mixture weight optimisation is carried out. In this example, it is held constant and set to $\eta = 0.1$. This means that the IS-n and IS-unif samplers can also be distinguished and we can compare their performance. Moreover, we set $\gamma = 0.5$.

The results of this simulation are visualised in Figure 2.4. Again, we see a close correspondence with Figure 2 of Daudel et al. (2022)[3]. Furthermore, we observe that the MG-IS-unif(γ) outperforms the other approaches in most cases.

Lastly, the paper scrutinised the choice of η. This has been done by plotting the final mixture weights $\boldsymbol{\lambda}_N$ of the MG-IS-unif(γ) algorithm under several settings. In this way, the effect of η on the mixture weights can be studied. The following values $\eta \in \{0., 0.05, 0.1, 0.5\}$ were used and the value of γ was set to $\gamma = 0.5$. The final weights are shown in Figure 2.5.

[3]Note that the plot for case (ii) and $J = 10$ has accidentally also been used for the plot of case (ii) and $J = 50$ in Daudel et al. (2022), therefore ours does not coincide.



(a) Case (i) and $J = 10$

(b) Case (i) and $J = 50$

(c) Case (ii) and $J = 10$

(d) Case (ii) and $J = 50$

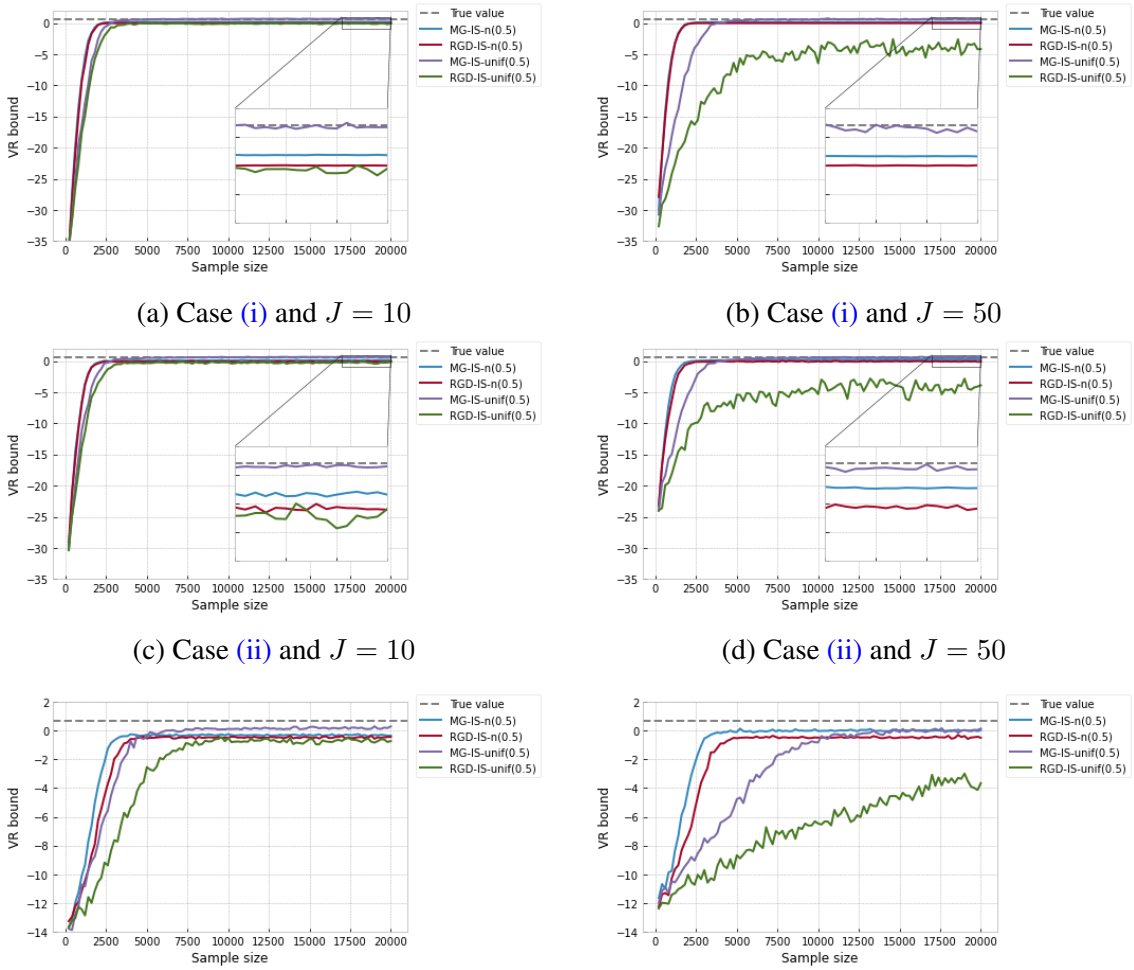
(e) Case (iii) and $J = 10$

(f) Case (iii) and $J = 50$

Figure 2.3: Replication of Figure 1 of Daudel et al. (2022). That is, the Monte Carlo estimate of the VR Bound averaged over 30 trials for the RGD and the MG approaches. Moreover, $\alpha = 0.2$, $d = 16$, $M = 200$, $\kappa_n = 0$, $\eta = 0$, and $q_n = \mu_n k$ at all times n and we examine the cases $\gamma \in \{0.1, 0.5, 1.\}$.

We observe a clear effect of η on the values of the mixture weights. For a high value of η, we even see that multiple weights get set to zero, thereby generating a mixture distribution consisting of less components. Hence, mixture weight optimisation does not only allow for more flexibility but also gives us a framework in which our variational density does not need to be needlessly complex. This clearly demonstrates the benefit of using this approach over more classical Variational Inference methods such as Li and Turner (2016) that do not consider mixture weight optimisation.

This completes the section on the replication of prior simulations performed by Daudel et al. (2022). We now finish this chapter by outlining some directions for further research.



(a) Case (i) and $J = 10$

(b) Case (i) and $J = 50$

(c) Case (ii) and $J = 10$

(d) Case (ii) and $J = 50$

(e) Case (iii) and $J = 10$

(f) Case (iii) and $J = 50$

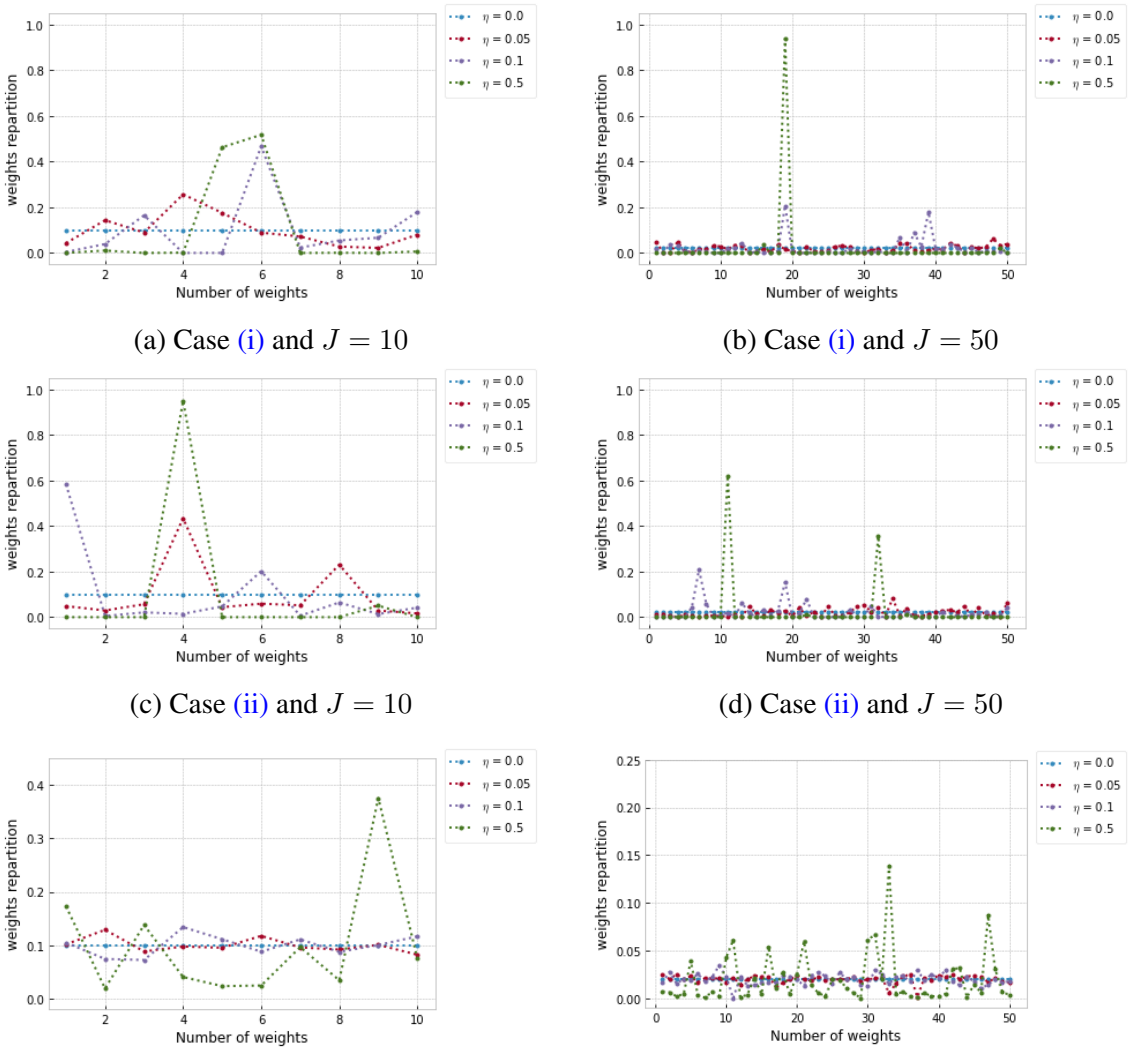
Figure 2.4: Replication of Figure 2 of Daudel et al. (2022). That is, the Monte Carlo estimate of the VR Bound averaged over 30 trials for the RGD and the MG approaches. Moreover, $\alpha = 0.2$, $d = 16$, $M = 200$, $\kappa_n = 0$, $\eta_n = 0.1$, and $\gamma = 0.5$.

2.3.4 Open directions of research

While the paper provides a novel framework for performing Variational Inference when using the α-divergence and a mixture model as variational distribution, we now detail two worthwhile directions for further research.

One limitation about the framework is that at the moment there do not exist convergence guarantees. The paper does prove that the method leads to a monotonic decrease in the α-divergence which provides some theoretical justification. However, this does not imply that the algorithm converges to the correct target distribution. Moreover, there are no known results about the rate of convergence.

Next to that, in the original paper, little guidance is provided on how to choose learning rate $\gamma_{j,n}$ in practice. It has been demonstrated in several experiments that a larger value of $\gamma_{j,n}$ leads to faster convergence. However, as shown by Figure 2.3, in some scenarios



(a) Case (i) and $J = 10$

(b) Case (i) and $J = 50$

(c) Case (ii) and $J = 10$

(d) Case (ii) and $J = 50$

(e) Case (iii) and $J = 10$

(f) Case (iii) and $J = 50$

Figure 2.5: Replication of Figure 3 of Daudel et al. (2022). That is, the final mixture weights $\boldsymbol{\lambda}_N$ of one run of the MG-IS-unif(0.5) algorithm. Moreover, $\alpha = 0.2$, $d = 16$, $M = 200$, $\gamma = 0.5$, and $\kappa_n = 0$ at all times n.

$\gamma_{j,n}$ can also be chosen too large and thereby deteriorate the convergence of the algorithm. Furthermore, in the experiments, the paper also only considers the case where $\gamma_{j,n} = \gamma$ is fixed. A more sophisticated approach where the learning rate is adaptable per iteration n and component j could lead to superior performances.

These limitations demonstrate that there are still multiple unexplored directions, both theoretical and empirical. Our aim is then to explore the directions mentioned above and we will start by establishing further theoretical contributions in the next chapter.

3

Theoretical contributions

In this chapter, we will study convergence of the algorithm when not using a mixture model, i.e. we set $J = 1$, as also outlined in Section 2.3.1. Below, we start by scrutinising the Gaussian case.

3.1 Study in the Gaussian case for $J = 1$

In this first section, we will derive several results about the optimal choice of γ_n and the corresponding convergence rate of the algorithm for a normal target density. We now briefly recall the Gaussian setting described in Example 1 and make a small adjustment to it.

Let $\mathsf{Y} = \mathbb{R}^d$ and ν be the Lebesgue measure. Moreover we let the kernel be Gaussian, that is, we set $k(\theta_n, y) = \mathcal{N}(y; m_n, \Sigma_n)$ where $\theta_n = (m_n, \Sigma_n) \in \mathsf{T}$. Here, T includes $\mathbb{R}^d$ and all *diagonal* positive definite symmetric matrices in $\mathbb{R}^{d\times d}$. Note the difference with Example 1, as we now restrict ourselves to diagonal matrices Σ_n whereas Example 1 employs a general covariance matrix Σ_n. Moreover, we again assume

$$0 < \int_{\mathsf{Y}} \left(1 + \|y\|^{2/(1-\alpha)}\right) p(y)\,\mathrm{d}y < \infty. \tag{3.1.1}$$

3.1.1 Maximisation approach

We will now present our first result, summarised in the following theorem. In this theorem, $c \geqslant 0$ is an unknown normalisation constant similarly as in Section 2.3.3 with the idea that we only know $p(y)$ up to a normalisation constant in Bayesian inference.

Theorem 4. *Consider the Gaussian setting just described with $d = 1$. Moreover, we assume that the target density is proportional to a Gaussian density $p(y) = c \times \mathcal{N}(y; m, \sigma^2)$*

where $m \in \mathbb{R}$ is the mean and $\sigma^2 > 0$ is the variance. For any choice of $(\gamma_n)_{n\geqslant 1}$ where $\gamma_n \in [\gamma_{\min}, 1]$ for all $n \geqslant 1$ and $\gamma_{\min} > 0$, the mean updating equation in (2.3.7) will converge with an exponential rate towards the optimal value, that is

$$\lim_{n\to\infty} m_n = m. \tag{3.1.2}$$

Proof. The assumption that $p(y)$ is proportional to a Gaussian density gives us the convenient result that $\breve{\varphi}_n^{(\alpha)}(y)$ is also Gaussian as

$$\begin{aligned}
\breve{\varphi}_n^{(\alpha)}(y) &\propto k(\theta_n, y)^\alpha p(y)^{1-\alpha} \\
&\propto \exp\left\{-\frac{\alpha}{2\sigma_n^2}(y-m_n)^2 - \frac{1-\alpha}{2\sigma^2}(y-m)^2\right\} \\
&\propto \exp\left\{-\frac{1}{2}\left(\frac{\alpha}{\sigma_n^2} + \frac{1-\alpha}{\sigma^2}\right) y^2 + \frac{\alpha}{\sigma_n^2} y m_n + \frac{1-\alpha}{\sigma^2} y m\right\} \\
&\propto \mathcal{N}(y; \widehat{m}_n, \widehat{\sigma}_n^2)
\end{aligned}$$

where

$$\widehat{m}_n = \alpha\frac{\widehat{\sigma}_n^2}{\sigma_n^2} m_n + (1-\alpha)\frac{\widehat{\sigma}_n^2}{\sigma^2} m = \frac{\alpha\sigma^2}{\alpha\sigma^2 + (1-\alpha)\sigma_n^2} m_n + \frac{(1-\alpha)\sigma_n^2}{\alpha\sigma^2 + (1-\alpha)\sigma_n^2} m, \tag{3.1.3}$$

$$\widehat{\sigma}_n^2 = \left(\frac{\alpha}{\sigma_n^2} + \frac{1-\alpha}{\sigma^2}\right)^{-1} = \frac{\sigma_n^2\sigma^2}{\alpha\sigma^2 + (1-\alpha)\sigma_n^2}. \tag{3.1.4}$$

Let us now consider the convergence of the sequence of means $(m_n)_{n\geqslant 1}$. In (3.1.3), we observe that $\widehat{m}_n$ is a convex combination of m_n and m. Now updating the mean according to (2.3.7) yields that

$$\begin{aligned}
m_{n+1} &= \gamma_n \widehat{m}_n + (1-\gamma_n)\, m_n \\
&= \gamma_n(1-\alpha)\frac{\widehat{\sigma}_n^2}{\sigma^2} m + \left(1 - \gamma_n\left(1 - \alpha\frac{\widehat{\sigma}_n^2}{\sigma_n^2}\right)\right) m_n.
\end{aligned}$$

Note by the definition $\widehat{\sigma}_n^2$ in (3.1.4) we have that $(1-\alpha)\,\frac{\widehat{\sigma}_n^2}{\sigma^2} = 1 - \alpha\frac{\widehat{\sigma}_n^2}{\sigma_n^2}$. Thus m_{n+1} is a convex combination of m and m_n and we can rewrite the mean update as

$$m_{n+1} = m_n + \gamma_n(1-\alpha)\frac{\widehat{\sigma}_n^2}{\sigma^2}(m - m_n) = m_n + a_n(m - m_n),$$

where we introduced the notation $a_n = \gamma_n(1-\alpha)\frac{\widehat{\sigma}_n^2}{\sigma^2} = \gamma_n\frac{(1-\alpha)\sigma_n^2}{\alpha\sigma^2 + (1-\alpha)\sigma_n^2}$. Clearly, $0 < a_n < 1\,\forall n \geqslant 1$ and hence $(m_n)_{n\geqslant 1}$ is a monotonic sequence. As shown in Lemma 1 in Appendix B.1, we can find a constant strictly positive lower bound on a_n which we denote by $a_{\min}$. Using this lower bound, we now define the sequence $(\tilde{m}_n)_{n\geqslant 1}$ satisfying

$$\tilde{m}_{n+1} = \tilde{m}_n + a_{\min}(m - \tilde{m}_n) \quad \text{for all } n \geqslant 1.$$

Now setting $\tilde{m}_1 = m_1$, it can be observed that $|m_n - m| \leqslant |\tilde{m}_n - m|$, which means that $(\tilde{m}_n)_{n\geqslant 1}$ moves slower towards m. Furthermore, it can easily be derived by induction that

$(\tilde{m}_n)_{n\geqslant 1}$ satisfies

$$\begin{aligned}\tilde{m}_n &= (1-a_{\min})^{n-1}m_1 + a_{\min}m\sum_{i=0}^{n-2}(1-a_{\min})^i \\ &= (1-a_{\min})^{n-1}\,m_1 + a_{\min}m\frac{1-(1-a_{\min})^{n-1}}{a_{\min}} \\ &= m + (1-a_{\min})^{n-1}\,(m_1-m).\end{aligned}$$

As $0 < 1-a_{\min} < 1$, we can prove the convergence of the sequence $(\tilde{m}_n)_{n\geqslant 1}$:

$$\lim_{n\to\infty}\tilde{m}_{n+1} = m + \lim_{n\to\infty}(1-a_{\min})^n\,(m_1-m) = m,$$

which converges at an exponential rate. Furthermore, we know that either $\tilde{m}_n \leqslant m_n \leqslant m$ or $\tilde{m}_n \geqslant m_n \geqslant m$ for all $n \geqslant 1$. By the Sandwich Theorem, it then follows that for the sequence of means $(m_n)_{n\geqslant 1}$, we also know that $\lim_{n\to\infty} m_n = m$ with an exponential rate. □

Before we comment on this result, let us also present the following corollary that establishes the convergence of the variance updates under $\gamma_n = \gamma = 1$ for all $n \geqslant 1$.

Corollary 1. *Consider the same setting as in Theorem 4. The sequence of means $(m_n)_{n\geqslant 1}$ converges fastest when $\gamma_n = \gamma = 1$ for all $n \geqslant 1$, i.e., choosing no regularisation. This leads to an exponential convergence rate of α. Moreover, this choice of $(\gamma_n)_{n\geqslant 1}$ also guarantees convergence of the variances update in (2.3.7), i.e.*

$$\lim_{n\to\infty}\sigma_n^2 = \sigma^2,$$

where σ_n^2 denotes the 1-dimensional equivalent of Σ_n. Similarly as with the mean updates, this converges exponentially at rate α.

Proof. As has been shown in the proof of Theorem 4, $(m_n)_{n\geqslant 1}$ is a monotonic sequence where a_n determines the speed of convergence. Using this, we can also derive the optimal regularisation sequence $(\gamma_n)_{n\geqslant 1}$. This follows from maximising a_n for each $n \geqslant 1$:

$$\max_{\gamma_n\in(0,1]} a_n = \max_{\gamma_n\in(0,1]} \gamma_n(1-\alpha)\frac{\widehat{\sigma}_n^2}{\sigma^2} = (1-\alpha)\frac{\widehat{\sigma}_n^2}{\sigma^2}. \tag{3.1.5}$$

Therefore, for each n, we set $\gamma_n = \gamma = 1$, i.e. the optimal sequence is constant and equal to 1. Let us derive the convergence rate of the sequence $(m_n)_{n\geqslant 1}$ under this choice of $(\gamma_n)_{n\geqslant 1}$. We infer that

$$\begin{aligned}\lim_{n\to\infty}\frac{|m_{n+1}-m|}{|m_n-m|} &= \lim_{n\to\infty}\frac{\left(1-\gamma_n\left(1-\alpha\frac{\widehat{\sigma}_{n+1}^2}{\sigma_{n+1}^2}\right)\right)^n}{\left(1-\gamma_n\left(1-\alpha\frac{\widehat{\sigma}_n^2}{\sigma_n^2}\right)\right)^{n-1}}, \\ &= \alpha\lim_{n\to\infty}\frac{\left(\frac{\widehat{\sigma}_{n+1}^2}{\sigma_{n+1}^2}\right)^n}{\left(\frac{\widehat{\sigma}_n^2}{\sigma_n^2}\right)^{n-1}} \\ &= \alpha,\end{aligned}$$

where the last step follows from Lemma 2 in Appendix B.2. Thus, $(m_n)_{n\geqslant 1}$ converges at exponential rate α.

We now move on to the updating sequence of $(\sigma_n^2)_{n\geqslant 1}$. Note that plugging $\gamma_n = \gamma = 1$ into (2.3.7) leads to following update for the variance

$$\sigma_{n+1}^2 = \widehat{\sigma}_n^2. \tag{3.1.6}$$

The exact expression for $\widehat{\sigma}_n^2$ can be found in (3.1.4). For this sequence $(\widehat{\sigma}_n^2)_{n\geqslant 1}$, we can prove that it converges. Namely, it can be shown using induction that it satisfies

$$\begin{aligned}\widehat{\sigma}_n^2 &= \frac{\sigma_1^2\sigma^2}{\alpha^n\sigma^2 + (1-\alpha)\sigma_1^2\sum_{i=0}^{n-1}\alpha^i}, \\ &= \frac{\sigma_1^2\sigma^2}{\sigma_1^2 + \alpha^n(\sigma^2 - \sigma_1^2)}.\end{aligned}$$

Since we know that $\alpha \in [0, 1)$, the convergence of $(\widehat{\sigma}_n^2)_{n\geqslant 1}$ trivially follows as

$$\lim_{n\to\infty}\widehat{\sigma}_n^2 = \frac{\sigma_1^2\sigma^2}{\sigma_1^2 + \lim_{n\to\infty}\alpha^n(\sigma^2 - \sigma_1^2)} = \sigma^2$$

which displays that the sequence $(\widehat{\sigma}_n^2)_{n\geqslant 1}$ converges with exponential speed. From (3.1.6), we then infer that $\sigma_n^2 \to \sigma^2$, also at exponential rate. Finally, we give corresponding convergence rate

$$\lim_{n\to\infty}\frac{|\sigma_{n+1}^2 - \sigma^2|}{|\sigma_n^2 - \sigma^2|} = \alpha\frac{\lim_{n\to\infty}\left|\frac{1}{\sigma_1^2+\alpha^n(\sigma^2-\sigma_1^2)}\right|}{\lim_{n\to\infty}\left|\frac{1}{\sigma_1^2+\alpha^{n-1}(\sigma^2-\sigma_1^2)}\right|} = \alpha, \tag{3.1.7}$$

where we used that for n large enough, both terms in absolute value must be strictly positive. □

Let us provide two remarks on the derived results. Firstly, the proportionality to α for the convergence rate can be explained when we look closer at $\varphi_n^{(\alpha)} = k(\theta_n, y)^\alpha p(y)^{1-\alpha}$ which can also be interpreted as a "proposal density". When α gets smaller, we see $\varphi_n^{(\alpha)}$ also starts to look more like $p(y)$. Therefore, a smaller α indicates more confidence. In this specific case, where $k(\theta_n, y)$ is chosen to be Gaussian and all expectations are analytically available, more confidence always leads to faster convergence. This case is however very rare in practice but it does provide us with further intuition about the proposed algorithm.

Secondly, the theorem above gives us some further intuition in the role of γ_n. When $\widehat{m}_n$ and $\widehat{\sigma}_n$ are analytically available and using the right variational density, we do not need any regularisation. However, in more complex scenarios, this will generally not be the case and we need to resort to stochastic approximations. We will get back to this, and the behaviour of γ_n in such a case, later in this dissertation. First, we mention that Theorem 4 can also be generalised to higher dimensions.

Theorem 5. *Consider the same Gaussian setting as already described where we now set* $p(y) = c \times \mathcal{N}(y; m, \Sigma)$ *where* Σ *is a general* $d \times d$ *diagonal matrix. For any choice of*

$(\gamma_n)_{n\geqslant 1}$ where $\gamma_n \in [\gamma_{\min}, 1]$ for all $n \geqslant 1$ and $\gamma_{\min} > 0$, the mean updating equation in (2.3.7) will converge at exponential speed towards the optimal value, that is

$$\lim_{n\to\infty} m_n = m. \tag{3.1.8}$$

Moreover, the optimal choice of $(\gamma_n)_{n\geqslant 1}$ is given by $\gamma_n = 1$ for all n, i.e., choosing no regularisation. This choice of $(\gamma_n)_{n\geqslant 1}$ also guarantees convergence of the covariances and leads to a convergence rate proportional to α for both sequences.

Proof. Recall that our variational density in this case is given by $k(\theta_n, y) = \mathcal{N}(y; m_n, \Sigma_n)$ where Σ_n is diagonal. As the covariance matrix is diagonal, $k(\theta_n, y)$ can also be written as a product of univariate Gaussian densities. Notice therefore, that we are implicitly assuming a Mean-field approximating family. This allows us to make apply Lemma 1 from Daudel et al. (2022) which states that within the Mean-field approximating family, the argmax problem from Theorem 1 can be solved component-wise. Because the target density $p(y)$ is also Gaussian and factorises, the component-wise problem boils down to the setting of Theorem 4. Hence, all results from Theorem 4 and Corollary 1 hold component-wise from which the desired result follows. An alternative, more involved, proof of this Theorem can be found in Appendix B.3. This proof follows the reasoning from Theorem 4 in a multi-dimensional setting. □

This concludes our first contribution to the framework from Daudel et al. (2022). In the next section, we demonstrate a link between our results and the existing literature.

3.1.2 Gradient-based approach

In this section, we will be using the gradient-based approach to show how the results in Theorem 4 and 5 relate to existing convergence results in the Gradient Descent literature. We do this by using the observation from the end of Section 2.3.1 that in the Gaussian setting, the mean updates from the maximisation approach can be interpreted as Gradient Descent steps for Rényi's α-divergence minimisation with a learning rate $r_n = \sigma^2(1-\alpha)\gamma_n$. Recall that the covariance was assumed to be fixed and known here, which we also assume throughout this section.

To demonstrate this, let us have a closer look at Gradient Descent while employing Rényi's α-divergence as the objective. Thus, by utilising the definition in (2.2.9), our objective is

$$g(m) = \mathbb{D}_\alpha^{(\mathrm{AR})}(K(m,\cdot)\|\mathbb{P}) = \frac{1}{\alpha(\alpha-1)} \log\left(\int_{\mathsf{Y}} k(m,y)^\alpha p(y|\mathcal{D})^{1-\alpha}\nu(\mathrm{d}y)\right). \tag{3.1.9}$$

By appendix A.2.2 of Daudel et al. (2022), (3.1.9) has a gradient given by

$$\nabla g(m_n) = \int_{\mathsf{Y}} \frac{\check{\varphi}_n^{(\alpha)}(y)}{\alpha-1} \frac{\partial \log k(m,y)}{\partial m}\bigg|_{(m,y)=(m_n,y)} \nu(\mathrm{d}y), \tag{3.1.10}$$

where we have set $p = p(\cdot, \mathcal{D})$. For the Gaussian setting from previous section, this

gradient is given by

$$\begin{aligned}\nabla g(m_n) &= \frac{1}{1-\alpha}\Sigma^{-1}\left(m_n - \int_{\mathsf{Y}} y\,\breve{\varphi}_n(y)\mathrm{d}y\right)\\ &= \frac{1}{1-\alpha}\Sigma^{-1}\left(m_n - \alpha m_n - (1-\alpha)m\right)\\ &= \Sigma^{-1}(m_n - m),\end{aligned}$$

where we used that $p(y)$ is proportional to a Gaussian density. Therefore the first moment of density $\breve{\varphi}$ is given by $\alpha m_n - (1-\alpha)m$ as also shown in (B.3.1) under known covariance matrices.

Let us now investigate the smoothness of objective g. One possible way to characterise β_n-smoothness is using the Hessian of the objective function (see Appendix B.4). This definition makes use of the notion of *Loewner order*. When a matrix A is greater or equal than B in Loewner order it is denoted by $A \succeq B$ and $A \preceq B$ vice versa. For the full definition, see the first part of Appendix B.3. In this example, it follows that the Hessian is given by

$$\nabla\nabla^T g(m_n) = \Sigma^{-1} = \frac{1}{\sigma^2}\boldsymbol{I_d}. \tag{3.1.11}$$

This shows that if we set $\beta_n = \beta = (1-\alpha)^{-1}\sigma^{-2}$, the objective g is β-smooth as

$$\nabla\nabla^T g(m_n) = \frac{1}{\sigma^2}\boldsymbol{I_d} \preceq \frac{1}{(1-\alpha)\sigma^2}\boldsymbol{I_d} = \beta\boldsymbol{I_d}, \quad \forall m_n \in \mathbb{R}^d$$

using that $\alpha \in [0,1)$. Notice that using this β is equivalent to using the β defined in Section 2.3.1 as the normalisation constant disappears since we are using $\breve{\varphi}_n^{(\alpha)}(y)$ in (3.1.10). For this choice of β, Theorem 2 holds and it is guaranteed that our objective monotonically decreases under these updates.

As these updates for the mean coincide with the mean updates from the maximisation approach, Theorems 4 and 5 still apply. We will now demonstrate how these results are related to the Gradient Descent literature. Firstly, note that in the Gaussian example, we have an explicit expression for the Hessian in (3.1.11) which is constant for all $m_n \in \mathbb{R}^d$. Therefore, we can easily deduce that g is not only β-smooth but also strongly convex (see Appendix B.4 for the definition). That is, we know that g is also σ^{-2}-strongly convex since

$$\nabla\nabla^T g(m) = \frac{1}{\sigma^2}\boldsymbol{I_d} \succeq \frac{1}{\sigma^2}\boldsymbol{I_d}, \quad \forall m_n \in \mathbb{R}^d.$$

From these smoothness and convexity results, the exponential decay seen in Theorems 4 and 5, also follows from the Gradient Descent literature. The following theorem can be found in Bubeck et al. (2015)[1]. Note that $\tilde{\alpha}$ is used here to denote the strong convexity index, not to be confused with α from the α-divergence.

Theorem 6 (Bubeck et al. (2015))**.** *Let $g : \mathbb{R}^d \mapsto \mathbb{R}$ be an objective function to be minimised. Assume this function is $\tilde{\alpha}$-strongly convex and β-smooth on $\mathbb{R}^d$. Then the*

[1]This is a slightly adjusted form of Theorem 3.10 from the book Bubeck et al. (2015). The book proves the constrained case whereas we show unconstrained case.

sequence $(m_n)_{n\geqslant 1}$ *of Gradient Descent iterations with constant learning rate* $r_n = r = \frac{1}{\beta}$ *satisfies for* $n \geqslant 1$,

$$\|m_{n+1} - m\|^2 \leqslant \left(1 - \frac{\tilde{\alpha}}{\beta}\right)^n \|m_n - m\|^2,$$

where m *denotes global minimiser of* g.

The proof can be found in Appendix B.5. Let us now demonstrate that g satisfies the assumptions of this theorem. Firstly, as derived above, g is σ^{-2}-strongly convex and β-smooth on parameter space $\mathbb{R}^d$. Next to that, the global minimiser of g is m since Rényi's α-divergence is zero if and only if $k(m, y) = p(y|\mathcal{D})$. Moreover, when setting $\gamma_n \equiv 1$, we are using learning rate

$$r_n = r = \sigma^2(1 - \alpha) = \frac{1}{\beta}.$$

Therefore, all conditions are satisfied and the theorem tells us that

$$\|m_{n+1} - m\|^2 \leqslant \left(1 - \frac{\sigma^{-2}}{(1-\alpha)^{-1}\sigma^{-2}}\right)^n \|m_1 - m\|^2 = \alpha^n \|m_1 - m\|^2. \tag{3.1.12}$$

Now note that Theorem 5 tells us that for constant Σ and $\gamma_n \equiv 1$, the maximisation approach updating schema satisfies

$$m_{n+1} - m = \alpha^n(m_1 - m).$$

We find that this implies the result given in (3.1.12) since

$$\begin{aligned} m_{n+1} - m = \alpha^n(m_1 - m) &\implies \|m_{n+1} - m\|^2 = \alpha^{2n} \|m_1 - m\|^2 \\ &\implies \|m_{n+1} - m\|^2 \leqslant \alpha^n \|m_1 - m\|^2. \end{aligned}$$

Hence, our results are in line with the results from Theorem 6 and we again see an exponential rate of convergence of α. In the terminology of Bubeck et al. (2015), the quantity $\frac{\beta}{\tilde{\alpha}} \geqslant 1$ which controls the rate of convergence is called the *condition number*. The closer this value is to 1, the faster the sequence converges. Therefore, in our example, the condition number is $(1-\alpha)^{-1}$ and a small α indeed leads to faster convergence. This concludes our results in the Gaussian setting, we will now move on to a more general setting.

3.2 Study in the exponential family case for $J = 1$

In this section, we will demonstrate the results under Gaussianity can also be generalised to the exponential family. First, let us give the definition of an exponential family distribution which we copy from Daudel et al. (2022).

Definition 3 (Exponential family)**.** *Let* E *be a Euclidean space endowed with an inner product* $\langle \cdot, \cdot \rangle_E$ *and its corresponding norm* $\| \cdot \|_E$, h *be a non-negative function defined on* Y *and* $S : \mathsf{Y} \to E$ *be a measurable function. In its canonical form, a member of the exponential family is defined by the parametric probability density*

$$k^{(o)}(\zeta, y) = h(y) \exp\left(\langle \zeta, S(y) \rangle_E - A(\zeta)\right), \quad y \in \mathsf{Y},\ \zeta \in E_0$$

where

$$E_0 := \left\{ \zeta \in E : \int_{\mathsf{Y}} h(y)\, \mathrm{e}^{\langle \zeta, S(y) \rangle_E}\, \nu(\mathrm{d}y) < \infty \right\}$$

and where the values taken by A on the subset E_0 are obtained from the normalising constraint $\int k^{(o)}(\zeta, \cdot)\mathrm{d}\nu = 1$. In this setting, ν is called the dominating measure and S the natural statistic. In what follows, $\operatorname{Int} E_0$ *denotes the interior of E_0.*

We also briefly review some key results that hold within the exponential family. Recall that in this family, S is a sufficient statistic, E_0 is a convex subset of E and A is infinitely differentiable in $\operatorname{Int} E_0$ (Casella et al., 2002). Furthermore, for all $\zeta \in E_0$, we know that

$$\begin{aligned} \mathbb{E}_\zeta[S(Y)] &= \nabla A(\zeta) \\ \operatorname{Var}_\zeta(S(Y)) &= \nabla \nabla^T A(\zeta), \end{aligned} \tag{3.2.1}$$

where $\mathbb{E}_\zeta$ and Var_ζ denote the expectation and covariance of S under the density $k^{(o)}(\zeta, \cdot)$, respectively. Let us now present the following regularity condition

(B1) The kernel density $k^{(o)}$ defined on $E_0 \times \mathsf{Y}$ is a member of the exponential family as in Definition 3. Moreover, we assume that:

(a) The function h on Y is positive.
(b) There is no affine hyperplane of E to which $S(y)$ belongs for ν-almost all $y \in \mathsf{Y}$.

This allows us to provide our first result in terms of the exponential family, in which we demonstrate that if the target density is proportional to the variational density, we can express the maximisation approach updating equations in terms of ∇A.

Theorem 7. *Assume (B1). Consider $p(y) = c \times f(y; \theta)$ and kernel $k(\theta_n, y) = f(y; \theta_n)$ where $f(y; \zeta) = h(y) \exp\{\langle \zeta, S(y) \rangle_E - A(\zeta)\}$ belongs to the exponential family. We then have a conjugacy relationship in which $\breve{\varphi}_n^{(\alpha)}(y) = f(y; \alpha\theta_n + (1-\alpha)\theta)$. Moreover, the updating equations following from the maximisation approach are given by:*

$$\nabla A(\theta_{n+1}) = \gamma_n \nabla A\left(\alpha\theta_n + (1-\alpha)\theta\right) + (1-\gamma_n)\nabla A\left(\theta_n\right). \tag{3.2.2}$$

Proof. For the target distribution, we know that $p(y) = c \times h(y) \exp\{\langle \theta, S(y) \rangle_E - A(\theta)\}$ and the variational density is given by $k(\theta_n, y) = h(y) \exp\{\langle \theta_n, S(y) \rangle_E - A(\theta_n)\}$. We then derive that

$$\begin{aligned} \breve{\varphi}_n^{(\alpha)}(y) &\propto k(\theta_n, y)^\alpha p(y)^{1-\alpha} \\ &\propto h(y) \exp\{\langle \alpha\theta_n + (1-\alpha)\theta, S(y) \rangle_E - \alpha A(\theta_n) - (1-\alpha)A(\theta)\} \\ &\propto h(y) \exp\{\langle \alpha\theta_n + (1-\alpha)\theta, S(y) \rangle_E\} \\ &\propto f(y; \alpha\theta_n + (1-\alpha)\theta). \end{aligned}$$

Furthermore, since assumption (B1) is satisfied, we can apply Theorem 4 from Daudel et al. (2022). This tells us that the corresponding argmax problem from Theorem 1 is solved by θ_{n+1} that satisfies

$$\nabla A(\theta_{n+1}) = \gamma_n \int_{\mathsf{Y}} S(y)\, \breve{\varphi}_n^{(\alpha)}(y)\, \nu(\mathrm{d}y) + (1-\gamma_n)\nabla A(\theta_n).$$

In this specific case, we know that $\breve{\varphi}_n^{(\alpha)}$ belongs to the same exponential family. Therefore, we conclude that the updating equations are:

$$\nabla A(\theta_{n+1}) = \gamma_n \nabla A\left(\alpha\theta_n + (1-\alpha)\theta\right) + (1-\gamma_n)\nabla A\left(\theta_n\right).$$

□

Theorem 7 shows that the algorithm possesses a conjugate relationship under the exponential family. Because of this relationship, we can express the updates in terms of ∇A. Due to (3.2.1), ∇A is in the literature occasionally referred to as the *mean mapping*. We can deduce further results from Theorem 7 by taking a closer look at this function. To do this, let us recall Proposition 3.2 from the book by Wainwright et al. (2008).

Proposition 5 (Wainwright et al. (2008))**.** *Let* $\mathcal{M} := \{\mu \in \mathbb{R}^d \mid \exists\, \zeta \in \operatorname{Int} E_0 \text{ s.t. } \mathbb{E}_\zeta[S(Y)] = \mu\}$. *The gradient mapping* $\nabla A : \operatorname{Int} E_0 \mapsto \mathcal{M}$ *is one-to-one if and only if the exponential representation is minimal.*

The exact definition of minimality of an exponential representation is not recalled here but the reader can assume that this is implied by (B1). Hence, Proposition 5 implies that the mean mapping is a bijective function. This assertion gives us some further insights into (3.2.2). This update tells us that θ_{n+1} has to be chosen such that

$$\mathbb{E}_{\theta_{n+1}}[S(Y)] = \gamma_n \nabla A(\alpha\theta_n + (1-\alpha)\theta) + (1-\gamma_n)\nabla A(\theta_n)$$

holds, i.e., the mean needs to be matched. Now because ∇A is bijective, we know there exists a unique θ_{n+1} that satisfies this. To illustrate the updates performed by (3.2.2), Figure 3.1 is provided.

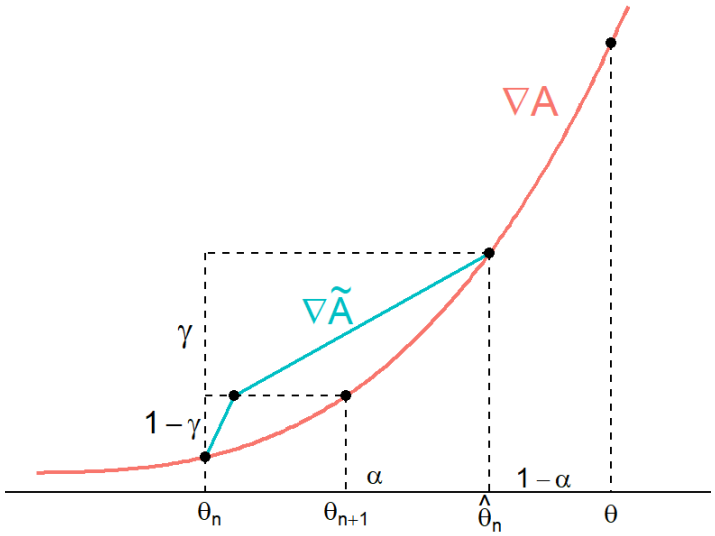


Figure 3.1: Illustration of update performed by (3.2.2) for an example function ∇A. Here $\hat{\theta}_n = \alpha\theta_n + (1-\alpha)\theta$. Next to that, $\nabla \tilde{A}$ is used to illustrate the proof of Theorem 8. Lastly, for this figure the values $\alpha = 0.3$ and $\gamma = 0.3$ were used.

Figure 3.1 will also help to illustrate the approach of the next result. In the following theorem, we generalise the results derived for the Gaussian updates to a 1-dimensional exponential family.

Theorem 8. *Assume (B1). Consider* $p(y) = c \times f(y;\theta)$ *and kernel* $k(\theta_n, y) = f(y;\theta_n)$ *where* $f(y;\zeta) = h(y)\exp\{\langle\zeta, S(y)\rangle_E - A(\zeta)\}$ *belongs to the exponential family. Moreover, assume* $d = 1$. *For any* $\theta_1 \in \operatorname{Int} E_0$ *and for any choice of* $(\gamma_n)_{n\geqslant 1}$ *where* $\gamma_n \in [\gamma_{\min}, 1]$ *for all* $n \geqslant 1$ *and* $\gamma_{\min} > 0$, *the updating equations in (3.2.2) will converge towards the optimal values, that is*

$$\lim_{n\to\infty} \theta_n = \theta.$$

Furthermore, notably, this converges at an exponential rate.

Proof. We know that ∇A is bijective, differentiable in $\operatorname{Int} E_0$, and A is convex. By the Mean Value Theorem, this implies that ∇A is strictly monotonically increasing. Moreover, we know that $\theta_1 \in \operatorname{Int} E_0$ and $\theta \in \operatorname{Int} E_0$. Using that $\operatorname{Int} E_0$ is convex, we obtain that $\alpha\theta_n + (1-\alpha)\theta \in \operatorname{Int} E_0$ and thus we can evaluate $\nabla A(\alpha\theta_n + (1-\alpha)\theta)$. Without loss of generality, assume that $\theta_1 < \theta$. Note that the interval $[\theta_1, \theta]$ is closed and all values in this interval are in $\operatorname{Int} E_0$ as this set is convex. Besides, ∇A is infinitely differentiable on this set, so its derivative continuous. Now by, the Extreme Value Theorem, it follows that there exists value m' and M' such that

$$m' \leqslant \nabla\nabla^T A(x) \leqslant M', \qquad \text{for all } x \in \operatorname{Int} E_0,$$

where $\nabla\nabla^T A$ denotes the second derivative of A to keep notation consistent with the multi-dimensional setting in Theorem 7. Note that $m' > 0$ (and trivially $M' > 0$) as ∇A is strictly monotonically increasing. Now, at iteration n, we will be moving to the θ_{n+1} that solves (3.2.2). Let us now give an expression that is smaller or equal to the step size $\theta_{n+1} - \theta_n$ for every n.

At iteration n, consider a function $\nabla\tilde{A}$ that moves from θ_n to $\alpha\theta_n + (1-\alpha)\theta$ with slope M' until it reaches $\gamma_n\nabla A(\alpha\theta_n + (1-\alpha)\theta) + (1-\gamma_n)\nabla A(\theta_n)$ and then it increases monotonically to the point $(\alpha\theta_n + (1-\alpha)\theta, \nabla A(\alpha\theta_n + (1-\alpha)\theta))$ as illustrated in Figure 3.1. This function always makes smaller steps than the sequence $(\theta_n)_{n\geqslant 1}$. In fact, as the function increases linearly with slope M', we can give an exact expression for this step size which we denote by s_n as we know that:

$$\begin{aligned} \nabla A(\theta_n) + s_n M' &= \gamma_n\nabla A(\alpha\theta_n + (1-\alpha)\theta) + (1-\gamma_n)\nabla A(\theta_n) \\ \iff \qquad s_n &= \frac{\gamma_n(\nabla A(\alpha\theta_n + (1-\alpha)\theta) - \nabla A(\theta_n))}{M'}. \end{aligned} \tag{3.2.3}$$

Furthermore, by the Mean Value Theorem, we know that

$$\frac{\nabla A(\alpha\theta_n + (1-\alpha)\theta) - \nabla A(\theta_n)}{\alpha\theta_n + (1-\alpha)\theta - \theta_n} = \frac{\nabla A(\alpha\theta_n + (1-\alpha)\theta) - \nabla A(\theta_n)}{(1-\alpha)(\theta - \theta_n)} \geqslant m',$$

which is again equivalent to

$$\nabla A(\alpha\theta_n + (1-\alpha)\theta) - \nabla A(\theta_n) \geqslant m'(1-\alpha)(\theta - \theta_n). \tag{3.2.4}$$

Now, (3.2.4) allows us to rewrite the find a lower bound on s_n that does not directly depend on ∇A. By plugging (3.2.4) into (3.2.3), we infer

$$s_n = \frac{\gamma_n(\nabla A(\alpha\theta_n + (1-\alpha)\theta) - \nabla A(\theta_n))}{M'} \geqslant \frac{m'}{M'}\gamma_n(1-\alpha)(\theta - \theta_n).$$

Now, as s_n is a lower bound on $\theta_{n+1} - \theta_n$, this tells us that

$$\begin{aligned}\theta_{n+1} - \theta_n &\geqslant \frac{m'}{M'}\gamma_n(1-\alpha)(\theta - \theta_n) \\ &\geqslant \frac{m'}{M'}\gamma_{\min}(1-\alpha)(\theta - \theta_n).\end{aligned}$$

Let us now denote $c = \frac{m'}{M'}\gamma_{\min}(1-\alpha)$ and note that $0 < c < 1$. We now introduce the sequence $(\tilde{\theta}_n)_{n\geqslant 1}$ satisfying

$$\tilde{\theta}_{n+1} = (1-c)\tilde{\theta}_n + c\theta, \qquad \text{for all } n \geqslant 1 \text{ and } \tilde{\theta}_1 = \theta_1.$$

By induction, we can easily derive that this sequence satisfies

$$\tilde{\theta}_{n+1} = \theta + (1-c)^n(\theta_1 - \theta).$$

From this, we can deduce that $\tilde{\theta}_n \to \theta$ at an exponential rate. Moreover, by induction, we can demonstrate that $\theta_n \geqslant \tilde{\theta}_n$ for all $n \geqslant 1$. Since we now have that $\tilde{\theta}_n \leqslant \theta_n \leqslant \theta$ for all $n \geqslant 1$, we conclude by the Sandwich Theorem that

$$\lim_{n\to\infty} \tilde{\theta}_n \leqslant \lim_{n\to\infty} \theta_n \leqslant \theta \implies \theta_n \to \theta.$$

As $\tilde{\theta}_n \to \theta$ at exponential speed, it now also follows that $(\theta_n)_{n\geqslant 1}$ converges at exponential speed. □

Theorem 8 gives rise to the following corollary that demonstrates that all results from Theorem 4 can be generalised to the exponential family.

Corollary 2. *In the setting of Theorem 8, the optimal choice of* $(\gamma_n)_{n\geqslant 1}$ *is given by* $\gamma_n = \gamma = 1$ *for all* $n \geqslant 1$. *This choice of* $(\gamma_n)_{n\geqslant 1}$ *yields a convergence rate proportional to* α.

Proof. As we know that $\theta_n \to \theta$, it follows from the continuity of ∇A that $(\nabla A(\theta_n))_{n\geqslant 1} \to \nabla A(\theta)$. Now, as ∇A is strictly monotonically increasing and $\alpha \in [0, 1)$, we know that either one of the following holds

$$\begin{aligned}&\text{(i)} \quad \nabla A(\theta_n) < \nabla A\left(\alpha\theta_n + (1-\alpha)\theta\right) \leqslant \nabla A(\theta) \text{ for every } n \geqslant 1, \\ &\text{(ii)} \quad \nabla A(\theta) \leqslant \nabla A\left(\alpha\theta_n + (1-\alpha)\theta\right) < \nabla A(\theta_n) \text{ for every } n \geqslant 1.\end{aligned} \tag{3.2.5}$$

Combining this with (3.2.2) and the fact that $\gamma_n \in (0, 1]$, it is easy to see that $(\nabla A(\theta_n))_{n\geqslant 1}$ converges monotonically. By the form of (3.2.2), we then see that it is optimal to choose $\gamma_n = 1$ at each iteration n. This would mean that we take the following updates

$$\nabla A(\theta_{n+1}) = \nabla A(\alpha\theta_n + (1-\alpha)\theta) \implies \theta_{n+1} = \alpha\theta_n + (1-\alpha)\theta, \tag{3.2.6}$$

where we used that ∇A is invertible. For this sequence, as was also done in the proof of 8, we can easily show that

$$\theta_n = \theta + \alpha^{n-1}(\theta_1 - \theta).$$

From this we immediately see that $(\theta_n)_{n\geqslant 1}$ converges to θ with exponential rate α. □

Corollary 2 demonstrates that when we use a variational density proportional to the target which belongs to the exponential family, it is again optimal to set $\gamma_n = \gamma = 1$ for all $n \geqslant 1$. With that, we conclude this section in which we focused on optimising $\Psi_\alpha(k(\theta, \cdot))$. That is, we focused on a single variational density and we derived convergence results in the Gaussian and exponential family case. It still remains unknown how these results generalise to a mixture model. In the next section, we illustrate the difficulties that occur in this case.

3.3 Extension to mixture models

Let us now focus on the original optimisation problem again, i.e., we concentrate on minimising $\Psi_\alpha(\mu_n k)$. In the previous section, we saw that the density function $\check{\varphi}_n^{(\alpha)}$ was crucial in the derivation of our results. This is due to the fact that all expectations were taken with respect to this density. Now, in the mixture case, we recall that this function also depends on mixture component j and is given by

$$\varphi_{j,n}^{(\alpha)}(y) = k(\theta_{j,n}, y) \left(\frac{\mu_n k(y)}{p(y)} \right)^{\alpha - 1}$$
$$\check{\varphi}_{j,n}^{(\alpha)} = \frac{\varphi_{j,n}^{(\alpha)}}{\int \varphi_{j,n}^{(\alpha)} \, \mathrm{d}\nu}.$$

We will now demonstrate that even in a simplified Gaussian case, we do not have closed-form expressions for the expectations with respect to this density.

Consider applying GMM optimisation as outlined in Algorithm 1 in a 1-dimensional setting with the Lebesgue measure. Moreover, we set $J = 2$ and fix the covariance matrix and mixture weights. In particular, we use the following the following variational density

$$\mu_n k(y) = \frac{1}{2}\mathcal{N}(y; m_{1,n}, 1) + \frac{1}{2}\mathcal{N}(y; m_{2,n}, 1)$$

and update the means following the MG approach $m_{j,n+1} = (1 - \gamma_n)\, m_{j,n} + \gamma_n \widehat{m}_{j,n}$. We furthermore assume that the target is proportional to a standard normal density $p(y) = c \times \mathcal{N}(y; 0, 1)$. Let us now inspect the form that $\varphi_{1,n}^{(\alpha)}(y)$ takes:

$$\begin{aligned}
\varphi_{1,n}^{(\alpha)}(y) &= \mathcal{N}(y; m_{1,n}, 1) \left(\frac{\frac{1}{2}\mathcal{N}(y; m_{1,n}, 1) + \frac{1}{2}\mathcal{N}(y; m_{2,n}, 1)}{c\mathcal{N}(y; 0, 1)} \right)^{\alpha - 1} \\
&\propto \mathcal{N}(y; m_{1,n}, 1) \left(\frac{\mathcal{N}(y; m_{1,n}, 1) + \mathcal{N}(y; m_{2,n}, 1)}{\mathcal{N}(y; 0, 1)} \right)^{\alpha - 1} \\
&\propto \exp\left\{ -\frac{1}{2}y^2 + y\, m_{1,n} \right\} \left(\exp\left\{ y\, m_{1,n} - \frac{1}{2} m_{1,n}^2 \right\} + \exp\left\{ y\, m_{2,n} - \frac{1}{2} m_{2,n}^2 \right\} \right)^{\alpha - 1}.
\end{aligned} \tag{3.3.1}$$

In this case, (3.3.1) is not proportional to a well-known distribution. Moreover, integrals of the form

$$\int_{\mathsf{Y}} \varphi_{1,n}^{(\alpha)}(y)\, g(y) \, \mathrm{d}y \tag{3.3.2}$$

for some function g, do not have a closed-form solution. So, to compute $\widehat{m}_{j,n}$, which is equivalent to using $g(y) = y$, we will need to resort to stochastic approximations as is also done in Algorithm 1. In this algorithm, the following approximation is employed

$$\widehat{m}_{j,n} = \frac{\sum_{m=1}^{M} \widehat{\varphi}_{j,n}^{(\alpha)}(Y_{m,n}) \cdot Y_{m,n}}{\sum_{m=1}^{M} \widehat{\varphi}_{j,n}^{(\alpha)}(Y_{m,n})}.$$

This estimator makes use of Normalised Importance Sampling (Kloek and Van Dijk, 1978; Geweke, 1989). Therefore, this estimator is known to be biased. Nevertheless, under particular conditions, it is still consistent.

Due to the lack of exact expressions and the stochasticity of the estimator, there is no theoretical justification for a particular choice of γ_n. To still obtain insights about the behaviour of the algorithm under different hyperparameters, we will perform several numerical experiments. These can be found in the next chapter.

4

Empirical contributions

This chapter consists of multiple numerical studies, which allow us to explore the algorithm in more complicated scenarios. In particular, we are interested in the behaviour of the algorithm when the target density is a mixture model. Before doing so, we validate the theoretical results from Chapter 3 empirically. Subsequently, we study the algorithm when the covariance updates are also implemented. Lastly, we look into adaptive learning rate schedulers.

4.1 Validation theoretical results

Let us consider the same setting as in Theorem 4 and 5. That is, we study the simple case where we have both a Gaussian target density and kernel. We found that choosing $\gamma_n = \gamma = 1$ for all $n \geqslant 1$ is the optimal learning rate for the mean updates. This led to an exponential convergence rate of α in both the mean and covariance parameters. In terms of the mean update in the 1-dimensional case, we found

$$m_n - m = \alpha^n \left(\frac{\hat{\sigma}_n^2}{\sigma_n^2} \right)^{n-1} (m_1 - m) \tag{4.1.1}$$

where in the limit, the convergence rate is simply α. In terms of logs, we expect that in the long term

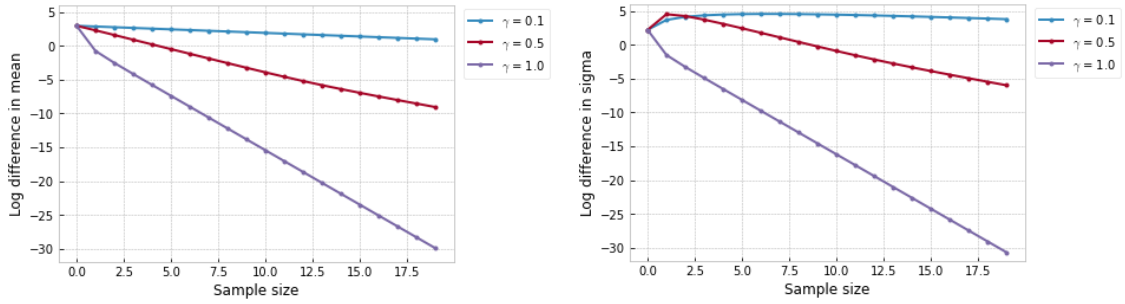
$$\log |m_n - m| \approx n \log(\alpha) + \log |m_1 - m|, \tag{4.1.2}$$

where we discard the fraction with covariances which converges to 1 by Lemma 2. Therefore, when plotting the log of the difference between m_n and its target m, we expect to see a linear trend with slope $\log(\alpha)$. In the following numerical experiment, we have implemented this to examine its validity. For the d-dimensional case with diagonal covariance matrices, (4.1.2) holds component-wise and we visualise one such component in the experiment.

Implementation details. In this experiment, we again use $\alpha = 0.2$ and $d = 16$ just as in the replication of Daudel et al. (2022) in Section 2.3.3. Since we are not using a mixture distribution, we have that $J = 1$ and the mixture weight parameters η_n and κ_n are therefore not applicable. For the initial values of $k(\theta_n, y) = \mathcal{N}(y; m_n, \Sigma_n)$, we set $m_1 = \mathbf{20}$ and $\Sigma_1 = 10\boldsymbol{I_d}$. For the target density, we use

$$p(y) = \mathcal{N}(y; m\boldsymbol{u_d}, \sigma^2 \boldsymbol{I_d}), \qquad \text{where } m = 1 \text{ and } \sigma^2 = 1.$$

In Figure 4.1, we demonstrate the convergence of the parameters corresponding to the first component, i.e. $m_{1,n}$ and $\sigma^2_{1,n}$, by showing the log differences with the corresponding target parameters m and σ^2. The figures for the other components look similar.



(a) $\log |m_{1,n} - m|$ against n (b) $\log |\sigma^2_{1,n} - \sigma^2|$ against n

Figure 4.1: Log plots of the absolute difference between mean and covariance and their respective targets in the Gaussian setting described in Chapter 3 for $\gamma \in \{0.1, 0.5, 1\}$. Moreover, $\alpha = 0.2$, $d = 16$, $M = 1$ (as the algorithm is deterministic), $\kappa_n = 0$ and $\eta_n = 0$ for all n.

Firstly, in Figure 4.1, we notice that setting $\gamma_n = \gamma = 1$ leads to faster convergence compared to the lower values of γ for both parameters. This is in line with the theoretical finding that $\gamma = 1$ is optimal for the mean update. Moreover, we clearly observe a linear trend which confirms the derived exponential convergence. We also observe that it takes a few iterations to establish this linear behaviour, which demonstrates the effect of the covariance fraction in (4.1.1) (and a similar term for the covariance updates which can be found in (3.1.7)). Nonetheless, it turns out that the approximation (4.1.2) is very accurate for larger values of n. When calculating the slope over the last 5 values in Figure 4.1a, we obtain a value of roughly -1.6093 whereas the approximation (4.1.2) yields $\log(\alpha) \approx -1.6094$. For the covariance updates, we observe similar accuracy. This validates the theoretical results derived in Chapter 3.

Next, we will study the effect of implementing covariance updates in Algorithm 1 and the effect of choosing a learning rate policy which is not constant.

4.2 New settings

In this section, we go beyond the constant variance and constant learning rate policy considered in the numerical experiments of Daudel et al. (2022) found in Section 2.3.3 in the

hope of getting improved empirical results. Let us start by implementing the covariance updates from Algorithm 1.

4.2.1 Covariance updates

Recall that the following update rule for the covariances can be found in Algorithm 1: for all $j = 1, \ldots, J$ and $n \geqslant 1$,

$$\Sigma_{j,n+1} = \gamma_n \widehat{\Sigma}_{j,n} + (1 - \gamma_n)\, \Sigma_{j,n} + \gamma_n\, (1 - \gamma_n)\, (\widehat{m}_{j,n} - m_{j,n})\, (\widehat{m}_{j,n} - m_{j,n})^T . \quad (4.2.1)$$

In the next experiments, we have included (4.2.1) in the algorithm, in addition to the MG and RGD updates for the mean. By also amending the covariances of each mixture component, more complex target distributions can be attained. For that reason, the extended algorithm has the potential to outperform the existing algorithm.

Firstly, we will consider the behaviour of the algorithm under the simple cases (i), (ii), and (iii) outlined in Section 2.3.3. In cases (i) and (ii), the covariances of the mixture components of the target are all set to $\boldsymbol{I_d}$. Hence, we do not expect that the algorithm including covariance updates necessarily outperforms the initial algorithm. However, it does provide a useful review of the extended algorithm under a simple base-case in which we compare it with the simpler algorithm. The results are plotted in Figure 4.2. Note that the covariance update algorithm is referred to as MA, which has been implemented using both the IS-n and IS-unif sampler.

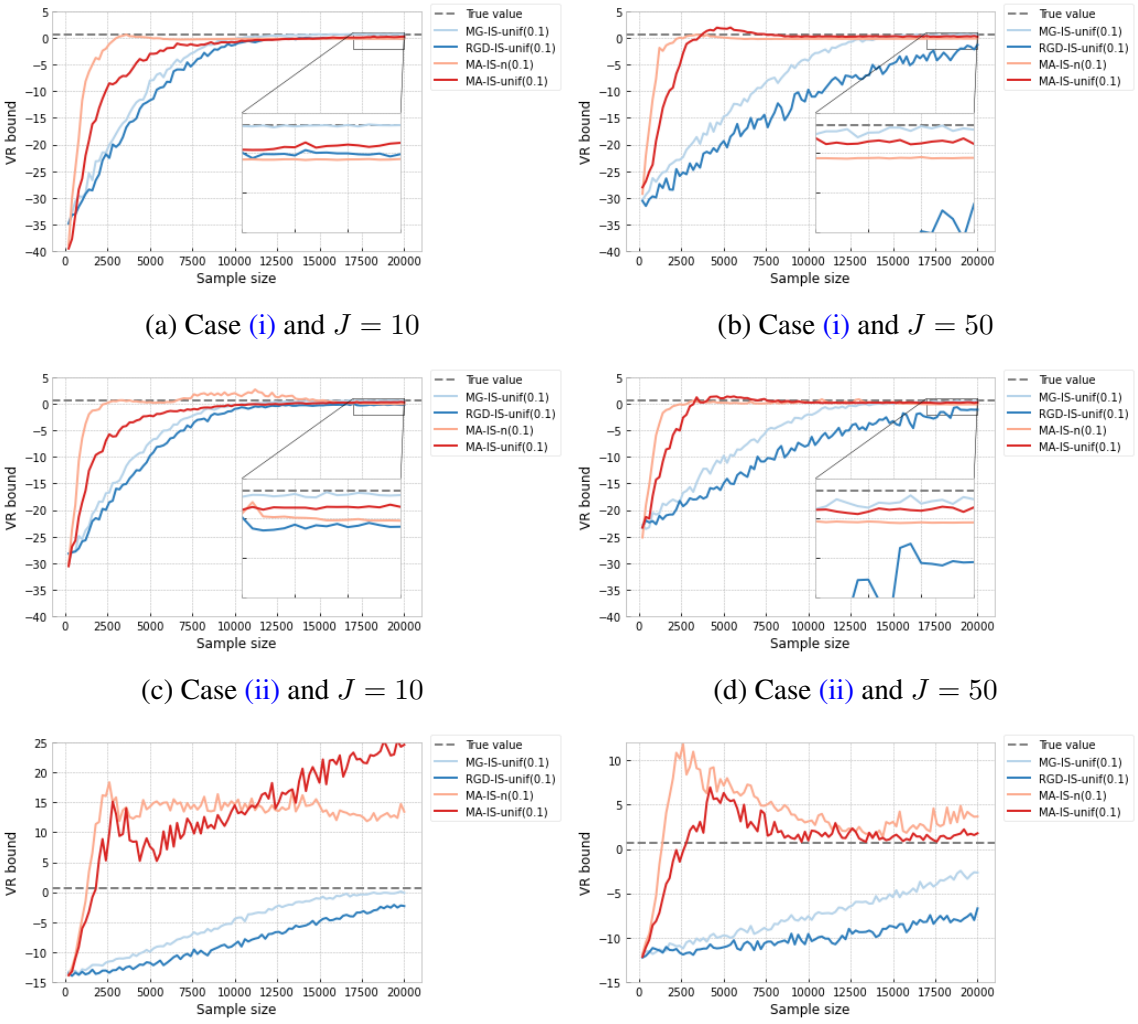
We observe in Figure 4.2 that this extended algorithm is an interesting alternative to the initial algorithm. For cases (i) and (ii) it converges quicker than the simpler algorithm. However, the MG-IS-unif(γ) algorithm does get closer to the true VR bound, which is likely caused by the use of correct covariance matrices. Next to that, observe in Figure 4.2e and 4.2f that the VR bounds of the MA algorithm are highly volatile and do not always converge. Especially for the $J = 10$ case, it highly overshoots the VR bound. This provides evidence that the algorithm including covariance updates is not robust.

In Figure C.1 in the appendix, a similar comparison under $\gamma = 0.5$ can be found. Here, we note that the MG and RGD approach without covariance updates give better performance in most cases. However, these approaches already made use of normal mixture components with the correct covariance matrix. In practice, this would rarely be the case. Therefore, to make a fairer comparison with the MA approach we now introduce a new case with unequal variances, inspired by case (ii).

(iv) *Imbalanced variance Gaussian Mixture Model.* The target p is a mixture density of three d-dimensional Gaussian distributions with unequal weights and unequal variances and multiplied by the positive constant c such that

$$p(y) = c \times \left[0.35\mathcal{N}\left(y; -2\boldsymbol{u_d}, 3\boldsymbol{I_d}\right) + 0.25\mathcal{N}\left(y; 2\boldsymbol{u_d}, 2\boldsymbol{I_d}\right) + 0.4\mathcal{N}\left(y; \boldsymbol{u_d}, 4\boldsymbol{I_d}\right)\right].$$

In Figure 4.3, we compare the behaviour of the extended algorithm with the initial algorithm under case (iv) for $\gamma \in \{0.1, 0.5\}$. Again, $c = 2$ is used.



(a) Case (i) and $J = 10$

(b) Case (i) and $J = 50$

(c) Case (ii) and $J = 10$

(d) Case (ii) and $J = 50$

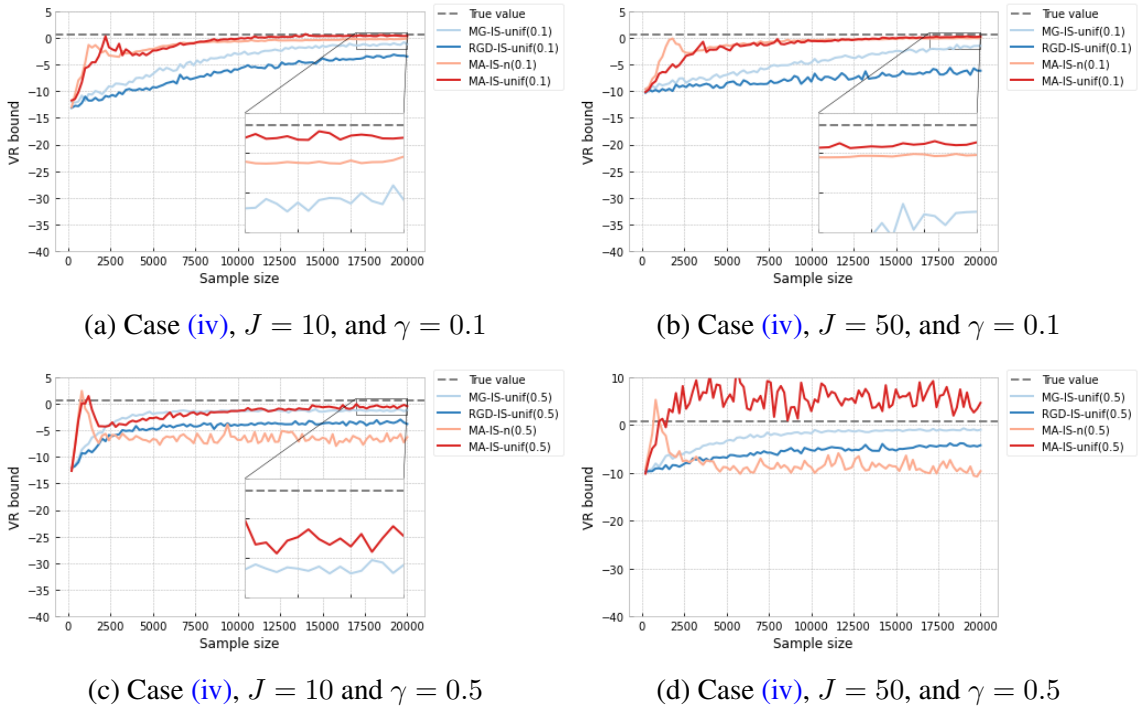
(e) Case (iii) and $J = 10$

(f) Case (iii) and $J = 50$

Figure 4.2: Implementation of Algorithm 1 including the covariance updates (referred to as MA) compared with the MG and RGD approach using the IS-unif sampler and $\gamma = 0.1$. Moreover, $\alpha = 0.2$, $d = 16$, $M = 200$, $\kappa_n = 0$ and $\eta_n = 0.1$ for all n.

Figure 4.3 provides us with some interesting insights. Firstly, for $\gamma = 0.1$, the MA algorithm substantially outperforms the initial algorithm. For the case where $J = 10$ and $\gamma = 0.5$, the performance of the MA-IS-unif(0.5) algorithm is comparable to that of the MG-IS-unif(0.5) algorithm. However, the version with covariance updates does get closer to the target in the end. This is likely caused by the fact that the true distribution is included in the variational family of the MA algorithm whereas it is not for MG approach. Lastly, we notice that the behaviour of the MA-IS-unif algorithm gets highly volatile again in Figure 4.3d as also seen in Figure C.1. For $\gamma = 1$ (not shown), the algorithm including covariance updates led to even more unstable results.

This demonstrates that the algorithm with covariance updates does have potential to outperform the initial algorithm, especially for a small value of γ. However, it is less robust than the simpler algorithm so further research should be done before it can be a practical alternative.



(a) Case (iv), $J = 10$, and $\gamma = 0.1$

(b) Case (iv), $J = 50$, and $\gamma = 0.1$

(c) Case (iv), $J = 10$ and $\gamma = 0.5$

(d) Case (iv), $J = 50$, and $\gamma = 0.5$

Figure 4.3: Implementation of Algorithm 1 including the covariance updates (referred to as MA) compared with the MG and RGD approach using the IS-unif sampler under case (iv) while $\gamma \in \{0.1, 0.5\}$. Moreover, $\alpha = 0.2$, $d = 16$, $M = 200$, $\kappa_n = 0$ and $\eta_n = 0.1$ for all n.

4.2.2 Adaptive learning rate policies

Recall that Daudel et al. (2022) allows $\gamma_{j,n}$ to be adjusted per iteration and component. Besides, Algorithm 1 focuses only on the case $\gamma_{j,n} = \gamma_n$. Yet, in the numerical experiments of the paper, the learning rate is also kept constant per iteration, i.e. $\gamma_n = \gamma$. In this section, we will perform several experiments in which $\gamma_{j,n}$ is adjusted per component and per iteration. Thereby, we aim to provide further intuition about the role of $\gamma_{j,n}$ and possibly improve the algorithm.

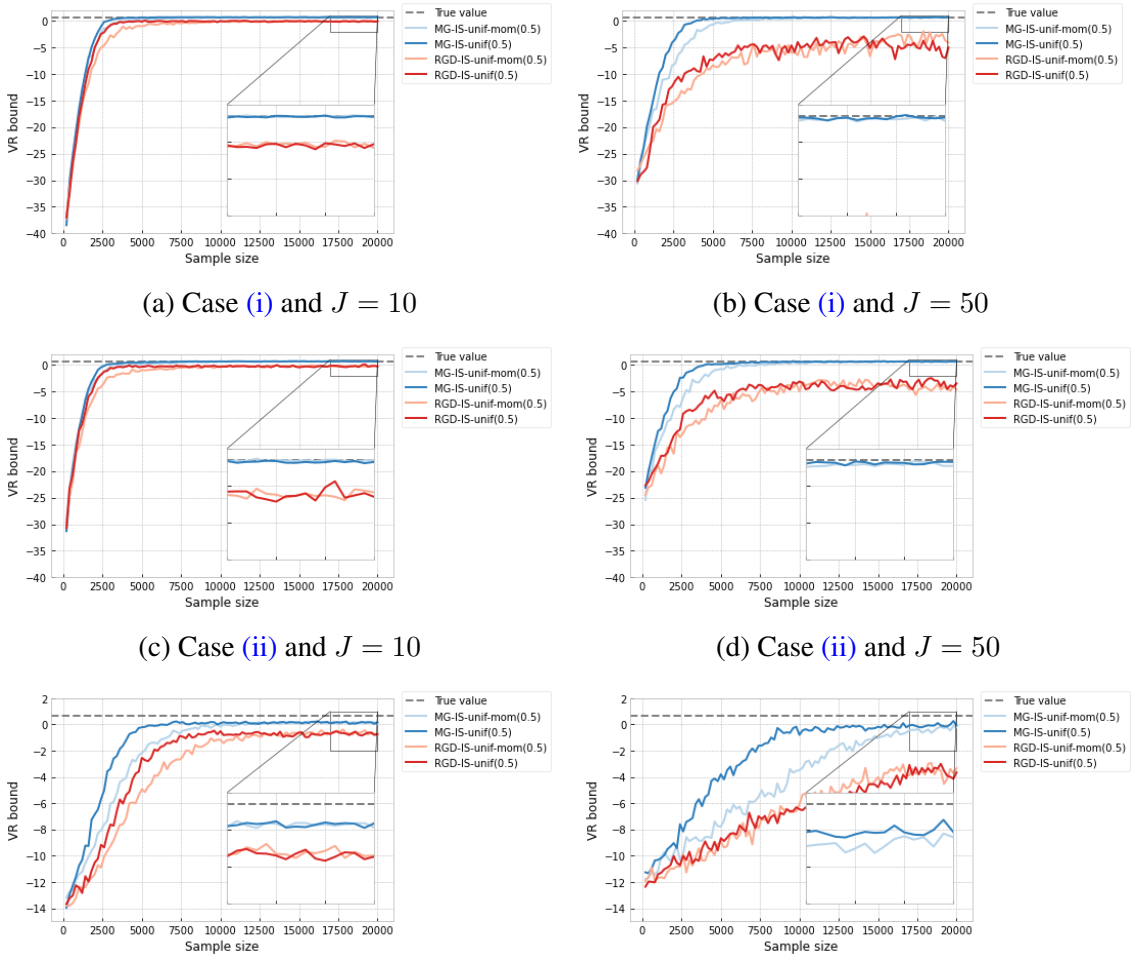
We will now explain and implement two methods that are common in the **S**tochastic **G**radient **D**escent (SGD) literature. The difference with Gradient Descent as demonstrated in (2.3.9) is that with SGD we use noisy gradient estimates $\widehat{\nabla g}$ of objective g. The reason for this is that generally these estimates are computationally cheaper to obtain. We then update the sequence $(\theta_n)_{n\geqslant 1}$ in the following way

$$\theta_{n+1} = \theta_n - r_n \widehat{\nabla g}(\theta_n). \tag{4.2.2}$$

However, what frequently happens is that the learning rate policy $(r_n)_{n\geqslant 1}$ is not chosen appropriately which leads to poor performance. We now introduce two methods that aim to adjust the step size r_n for every $n \geqslant 1$ such that the method converges faster.

Momentum

Let us first introduce the concept of momentum, a common method when training deep neural networks (Polyak, 1964; Sutskever et al., 2013). The idea here is that we average



(a) Case (i) and $J = 10$

(b) Case (i) and $J = 50$

(c) Case (ii) and $J = 10$

(d) Case (ii) and $J = 50$

(e) Case (iii) and $J = 10$

(f) Case (iii) and $J = 50$

Figure 4.4: Implementation of momentum for the maximisation approach and RGD approach using the IS-unif sampler and $\gamma = 0.5$. Moreover, $\alpha = 0.2$, $d = 16$, $M = 200$, $\kappa_n = 0$ and $\eta_n = 0.1$ for all n.

our noisy gradient estimates $\widehat{\nabla g}$ out over previous minibatches in the following way

$$\Delta_{n+1} = \beta\Delta_n + (1-\beta)\,\widehat{\nabla g}\,(\theta_n)$$
$$\theta_{n+1} = \theta_n - r_n\Delta_{n+1}.$$

Here $0 < \beta < 1$ is a hyperparameter indicating the weight of previous gradients, i.e. the amount of history that is used. Momentum can be very useful in narrow valleys where the direction along the valley has a small gradient but can be accumulated over time. In this way, momentum keeps track of its 'velocity', whereas vanilla SGD halts after each iteration and then obtains a new estimate.

The results from implementing momentum can be found in Figure 4.4. We observe that there are no cases where the new algorithm outperforms the initial algorithm. It performs roughly as well or it behaves worse.

Because plain momentum does not seem to improve the algorithm in these cases, we will implement an enhanced technique that makes use of momentum.

ADAM

One of the most popular learning rate schedulers in the literature at the moment is ADAM (Kingma and Ba, 2014). It stands for **ADA**ptive **M**oment estimation and is based on using the concepts of momentum and different learning rates for different parameters.

We have already seen how momentum can improve SGD. However, another limiting factor in vanilla SGD is that it uses a single step size for each component. Due to the nonlinearity of the objective, different components might have different scales which motivates the use of different step sizes. ADAM allows for this by not only keeping track of the gradient but also scaling the step size based on the squared gradient. The scheme is demonstrated below:

$$\begin{aligned}
\Delta_{n+1} &= \beta_1 \Delta_n + (1-\beta_1)\,\nabla g\left(\theta_n\right) \\
\tilde{\Delta}_{n+1} &= \frac{\Delta_{n+1}}{1-\beta_1^{n+1}} \\
V_{n+1} &= \beta_2 V_n + (1-\beta_2)\left(\nabla g\left(\theta_n\right)\right)^2 \\
\tilde{V}_{n+1} &= \frac{V_{n+1}}{1-\beta_2^{n+1}} \\
\theta_{n+1} &= \theta_n - \frac{r_n}{\sqrt{\tilde{V}_{n+1}+\epsilon}}\tilde{\Delta}_{n+1}.
\end{aligned}$$

Here, the divison by $1-\beta^{n+1}$ accounts for the fact that earlier steps are averaging over fewer gradients. Usually, as also indicated by Kingma and Ba (2014), setting $\beta_1 = 0.99$, $\beta_2 = 0.999$ and $\epsilon = 1 \times 10^{-8}$ tends to work quite well in practice.

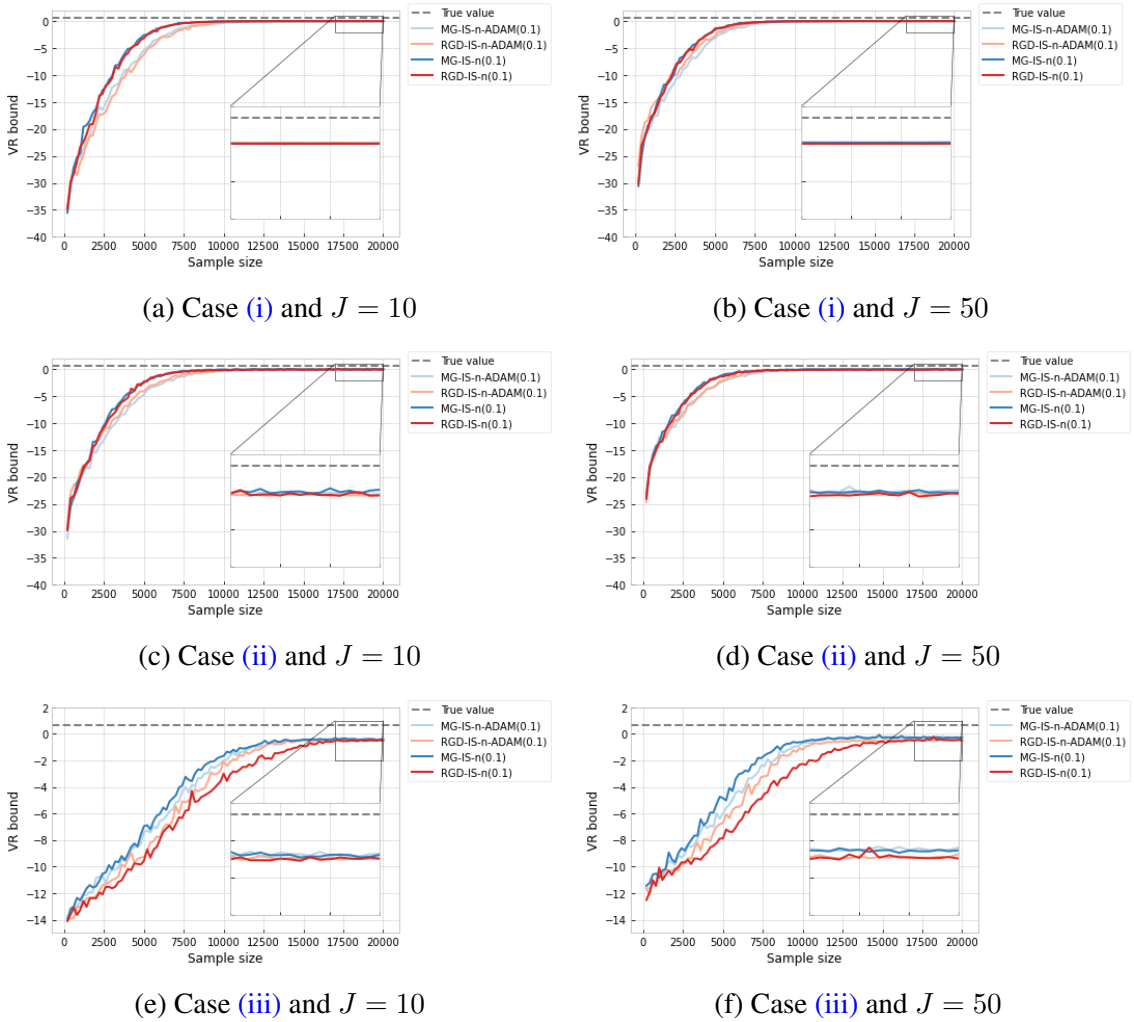
In terms of the MG update for the Gaussian mean, we note that we can rewrite this to

$$m_{j,n+1} = (1-\gamma_n)\, m_{j,n} + \gamma_n \widehat{m}_{j,n} = m_{j,n} - \gamma_n \left(m_{j,n} - \widehat{m}_{j,n}\right).$$

Here, we recognise the standard gradient descent step (2.3.9) where the typical gradient is now given by the difference $m_{j,n} - \widehat{m}_{j,n}$. Therefore, a possible approach is to apply ADAM here which leads to the following updating scheme

$$\begin{aligned}
\Delta_{n+1} &= \beta_1 \Delta_n + (1-\beta_1)\left(m_n - \widehat{m}_n\right) \\
\tilde{\Delta}_{n+1} &= \frac{\Delta_{n+1}}{1-(\beta_1)^{n+1}} \\
V_{n+1} &= \beta_2 V_n + (1-\beta_2)\left(m_n - \widehat{m}_n\right)^2 \\
\tilde{V}_{n+1} &= \frac{V_{n+1}}{1-(\beta_2)^{n+1}} \\
m_{n+1} &= m_n - \frac{\gamma_n}{\sqrt{\tilde{V}_{n+1}+\epsilon}}\tilde{\Delta}_{n+1},
\end{aligned}$$

where $(\Delta_n)_{n\geqslant 1}$ and $(V_n)_{n\geqslant 1}$ are sequences of vectors and the squared, square root and division operators are applied element-wise. Furthermore, we can equally apply this to the RGD approach which is already in the right form.



(a) Case (i) and $J = 10$

(b) Case (i) and $J = 50$

(c) Case (ii) and $J = 10$

(d) Case (ii) and $J = 50$

(e) Case (iii) and $J = 10$

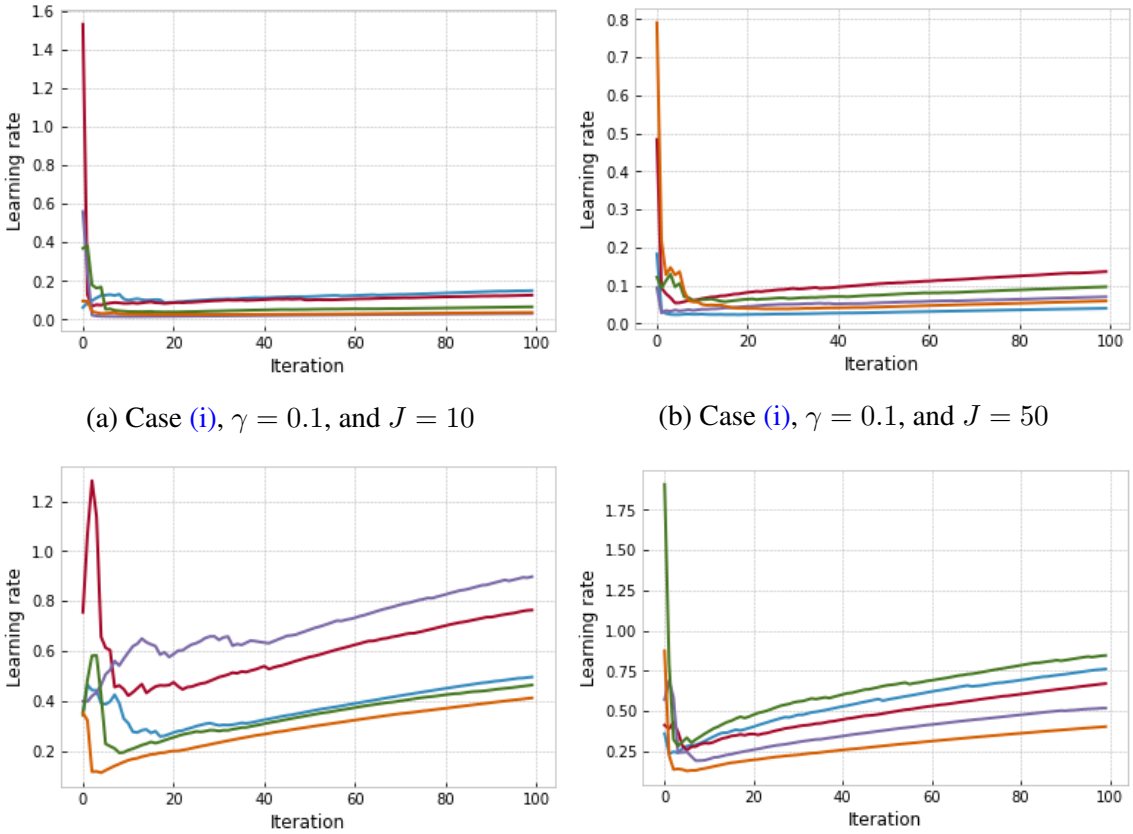
(f) Case (iii) and $J = 50$

Figure 4.5: Implementation of ADAM for the maximisation approach and RGD approach using the IS-n sampler and $\gamma = 0.1$. Moreover, $\alpha = 0.2$, $d = 16$, $M = 200$, $\kappa_n = 0$ and $\eta_n = 0.1$ for all n.

To examine the performance of this extended algorithm, several numerical experiments were performed. In general the conclusion was that ADAM did not improve on the MG approach but it did in a few cases for the RGD approach. One such example can be found in Figure 4.5. Here, we used $\gamma = 0.1$ and the IS-n sampler. It should be noted that for the MG approach, ADAM leads in general to equal or slower convergence rates. For the RGD approach, we observe that ADAM can in fact improve this algorithm in terms of case (iii) where we observe faster convergence.

To better understand the difference between both approaches and the validity of ADAM, we have visualised the behaviour of the learning rate of one component in the MG-IS-n-ADAM(γ) approach for multiple replications in Figure 4.6. This figure immediately displays that the learning rate can now become larger than 1 whereas this was not allowed in the original framework. Thus, the monotonic decrease in Ψ_α established by Daudel et al. (2022) is not guaranteed anymore. Next to that, Figure 4.6 also elucidates the comparable performance of the MG-IS-n(γ) approach with and without ADAM. In general,

the learning rate in Figure 4.6 is still close to the constant learning rate used in the MG-IS-n(γ) approach, that is $\gamma = 0.1$ or $\gamma = 0.5$. In the initial stage, there can be some large fluctuations in learning rate but after some time the approach keeps getting more confident and learning rate keeps growing. This is explained by the fact that we divide by $\sqrt{(m_n - \widehat{m}_n)^2} + \epsilon$, i.e. a measure of how close we are.



(a) Case (i), $\gamma = 0.1$, and $J = 10$

(b) Case (i), $\gamma = 0.1$, and $J = 50$

(c) Case (i), $\gamma = 0.5$, and $J = 10$

(d) Case (i), $\gamma = 0.5$, and $J = 50$

Figure 4.6: Learning rate corresponding to the first component in the MG-IS-n-ADAM(γ) approach for 5 different replications of the algorithm with $N = 100$ iterations. Moreover, $\alpha = 0.2$, $d = 16$, $M = 200$, $\kappa_n = 0$ and $\eta_n = 0.1$ for all n.

Further results are included in Appendix C.3 where the IS-unif sampler and $\gamma = 0.5$ is used. For this setting, we note that implementing ADAM does not advance the convergence rate of the algorithm. Next to that, the corresponding learning rates for the IS-unif sampler and $\gamma \in \{0.1, 0.5\}$ are included in Appendix C.4 which follow similar patterns as the learning rates in Figure 4.6.

5

Conclusion

In this dissertation, we focused on the family of iterative algorithms introduced by Daudel et al. (2022). This novel method offers a flexible framework that minimises the α-divergence while using a mixture model as variational density. Thereby, it has the potential to capture complex multimodal target densities. Nevertheless, this method has only recently been proposed and many aspects have not been fully investigated. Therefore, the goal of this dissertation was to provide further understanding of this technique.

After having introduced all the required background, we took a theoretical perspective and studied the method in a simplified setting. That is, we assumed a Gaussian target density and additionally used a Gaussian density as our variational density. In this case, there is a conjugate relationship in the model which allowed us to derive analytic expressions for the updates. Consequently, we determined the optimal learning rate policy $(\gamma_n)_{n\geqslant 1}$ and the corresponding convergence rate, which turned out to be exponential. Moreover, we could relate these results to the existing Gradient Descent literature.

On top of this, we extended these results under Gaussianity to the exponential family. That is, we assumed a target and variational density from the same exponential family. Here, the same conjugate relationship was observed, which again led to analytic expression for the updates. Moreover, the convergence and optimality of using $\gamma_n = 1$ for all $n \geqslant 1$ was derived. These results provided further theoretical justification for the algorithm.

Subsequently, the derived results were verified in a numerical experiment. Furthermore, due to the lack of analytically available updating equations in the mixture case, the algorithm was also simulated in a mixture setting. In particular, the covariance updates given in the GMM algorithm of Daudel et al. (2022) were implemented. This displayed that these updates can significantly improve the algorithm, whereas it also led to more volatility and it therefore proved to be a less robust approach. Besides, we made an empirical

contribution by investigating an adjustable learning rate policy $\gamma_{j,n}$. This was done by implementing the well-established techniques of momentum and ADAM. However, these techniques did not improve the algorithm in most experiments.

This provided us with further understanding of the algorithm. However, many facets still remain unknown. In terms of further research, the following directions could be pursued. In this dissertation, we assumed that $\alpha \in [0, 1)$, future work might consider generalising the existing results for a broader range of α. Nonetheless, even for a broader range of α, little is known about which value of α works best in a specific application. Advances could be made by providing more intuition to the practitioner on how to choose this hyperparameter. More than that, the approach could be enhanced by including a technique to automatically tune this hyperparameter α such as is done in Wang et al. (2018). Furthermore, additional work on learning rate $\gamma_{j,n}$ could be advantageous. On a theoretical level, the optimality of this sequence in alternative scenarios could be derived. On a practical level, more advanced learning rate schedulers should also be profitable. Lastly, more sophisticated Monte Carlo techniques are expected to yield improved performance in terms of estimating the intractable integrals appearing in the algorithm.

To conclude it all, we can say that the algorithm provided by Daudel et al. (2022) is a promising technique. We provided theoretical evidence of convergence in some simplified settings, and empirically it has been shown to outperform other existing algorithms. However, several aspect still remain unknown and further work should be done to enable applications to real-world Machine Learning problems.

Bibliography

Alquier, Pierre and James Ridgway (2020). Concentration of tempered posteriors and of their variational approximations. The Annals of Statistics 48(3), 1475–1497.

Alquier, Pierre, James Ridgway, and Nicolas Chopin (2016). On the properties of variational approximations of gibbs posteriors. The Journal of Machine Learning Research 17(1), 8374–8414.

Bhatia, Rajendra (2013). Matrix analysis, Volume 169. Springer Science & Business Media.

Bishop, Christopher M (2006). Pattern recognition and machine learning, Volume 4. Springer.

Blei, David M and Michael I Jordan (2006). Variational inference for dirichlet process mixtures. Bayesian analysis 1(1), 121–143.

Blei, David M, Alp Kucukelbir, and Jon D McAuliffe (2017). Variational inference: A review for statisticians. Journal of the American statistical Association 112(518), 859–877.

Blei, David M, Andrew Y Ng, and Michael I Jordan (2003). Latent dirichlet allocation. Journal of machine Learning research 3(Jan), 993–1022.

Blundell, Charles, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra (2015). Weight uncertainty in neural network. In International conference on machine learning, pp. 1613–1622. PMLR.

Bubeck, Sébastien et al. (2015). Convex optimization: Algorithms and complexity. Foundations and Trends® in Machine Learning 8(3-4), 231–357.

Casella, G., R.L. Berger, and Brooks/Cole Publishing Company (2002). Statistical Inference. Duxbury advanced series in statistics and decision sciences. Thomson Learning.

Cauchy, Augustin et al. (1847). Méthode générale pour la résolution des systemes d'équations simultanées. Comp. Rend. Sci. Paris 25(1847), 536–538.

Cichocki, Andrzej and Shun-ichi Amari (2010). Families of alpha-beta-and gamma-divergences: Flexible and robust measures of similarities. Entropy 12(6), 1532–1568.

Csiszár, Imre (1964). Eine informationstheoretische ungleichung und ihre anwendung auf

beweis der ergodizitaet von markoffschen ketten. Magyer Tud. Akad. Mat. Kutato Int. Koezl. 8, 85–108.

Dagum, Paul and Michael Luby (1993). Approximating probabilistic inference in bayesian belief networks is np-hard. Artificial intelligence 60(1), 141–153.

Daudel, Kamélia and Randal Douc (2021). Mixture weights optimisation for alpha-divergence variational inference. Advances in Neural Information Processing Systems 34, 4397–4408.

Daudel, Kamélia, Randal Douc, and François Portier (2021). Infinite-dimensional gradient-based descent for alpha-divergence minimisation. The Annals of Statistics 49(4), 2250–2270.

Daudel, Kamélia, Randal Douc, and François Roueff (2022). Monotonic alpha-divergence minimisation. arXiv preprint arXiv:2103.05684.

Dieng, Adji Bousso, Dustin Tran, Rajesh Ranganath, John Paisley, and David Blei (2017). Variational inference via χ upper bound minimization. Advances in Neural Information Processing Systems 30.

Doucet, Arnaud, Nando de Freitas, and Neil Gordon (2001). An introduction to sequential monte carlo methods. In Sequential Monte Carlo methods in practice, pp. 3–14. Springer.

Gal, Yarin and Zoubin Ghahramani (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In international conference on machine learning, pp. 1050–1059. PMLR.

Geweke, John (1989). Bayesian inference in econometric models using monte carlo integration. Econometrica: Journal of the Econometric Society, 1317–1339.

Hellinger, Ernst (1909). Neue begründung der theorie quadratischer formen von unendlichvielen veränderlichen. Journal für die reine und angewandte Mathematik 1909(136), 210–271.

Hernandez-Lobato, Jose, Yingzhen Li, Mark Rowland, Thang Bui, Daniel Hernández-Lobato, and Richard Turner (2016). Black-box alpha divergence minimization. In International Conference on Machine Learning, pp. 1511–1520. PMLR.

Hubbard, John H and Barbara Burke Hubbard (2015). Vector calculus, linear algebra, and differential forms: a unified approach. Matrix Editions.

Jordan, Michael I, Zoubin Ghahramani, Tommi S Jaakkola, and Lawrence K Saul (1998). An introduction to variational methods for graphical models. In Learning in graphical models, pp. 105–161. Springer.

Kingma, Diederik P and Jimmy Ba (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Kingma, Diederik P and Max Welling (2013). Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114.

Kloek, Tuen and Herman K Van Dijk (1978). Bayesian estimates of equation system parameters: an application of integration by monte carlo. Econometrica: Journal of the Econometric Society, 1–19.

Kuleshov, Volodymyr and Stefano Ermon (2017). Neural variational inference and learning in undirected graphical models. Advances in Neural Information Processing Systems 30.

Kullback, Solomon and Richard A Leibler (1951). On information and sufficiency. The annals of mathematical statistics 22(1), 79–86.

Li, Yingzhen, José Miguel Hernández-Lobato, and Richard E Turner (2015). Stochastic expectation propagation. Advances in neural information processing systems 28.

Li, Yingzhen and Richard E Turner (2016). Rényi divergence variational inference. Advances in neural information processing systems 29.

Lieb, Elliott H and Michael Loss (2001). Analysis, Volume 14. American Mathematical Soc.

Lindsay, Bruce G (1994). Efficiency versus robustness: the case for minimum hellinger distance and related methods. The annals of statistics 22(2), 1081–1114.

Meenakshi, AR and C Rajian (1999). On a product of positive semidefinite matrices. Linear algebra and its applications 295(1-3), 3–6.

Minka, Thomas (2004). Power ep. Technical report, Technical report, Microsoft Research, Cambridge.

Minka, Tom et al. (2005). Divergence measures and message passing. Technical report, Technical report, Microsoft Research.

Minka, Thomas P (2013). Expectation propagation for approximate bayesian inference. arXiv preprint arXiv:1301.2294.

Morimoto, Tetsuzo (1963). Markov processes and the h-theorem. Journal of the Physical Society of Japan 18(3), 328–331.

Neal, Radford M (1993). Probabilistic inference using Markov chain Monte Carlo methods. Department of Computer Science, University of Toronto Toronto, ON, Canada.

Neal, Radford M (2012). Bayesian learning for neural networks, Volume 118. Springer Science & Business Media.

Parisi, Giorgio and Ramamurti Shankar (1988). Statistical field theory.

Polyak, Boris T (1964). Some methods of speeding up the convergence of iteration methods. Ussr computational mathematics and mathematical physics 4(5), 1–17.

Rényi, Alfréd et al. (1961). On measures of entropy and information. In Proceedings of the fourth Berkeley symposium on mathematical statistics and probability, Volume 1. Berkeley, California, USA.

Robert, Christian P and George Casella (1999). Monte Carlo statistical methods, Volume 2. Springer.

Sutskever, Ilya, James Martens, George Dahl, and Geoffrey Hinton (2013). On the importance of initialization and momentum in deep learning. In International conference on machine learning, pp. 1139–1147. PMLR.

Van Erven, Tim and Peter Harremos (2014). Rényi divergence and kullback-leibler divergence. IEEE Transactions on Information Theory 60(7), 3797–3820.

Vial, Jean-Philippe (1982). Strong convexity of sets and functions. Journal of Mathematical Economics 9(1-2), 187–205.

Wainwright, Martin J, Michael I Jordan, et al. (2008). Graphical models, exponential families, and variational inference. Foundations and Trends® in Machine Learning 1(1–2), 1–305.

Wang, Dilin, Hao Liu, and Qiang Liu (2018). Variational inference with tail-adaptive f-divergence. Advances in Neural Information Processing Systems 31.

Zhang, Cheng, Judith Bütepage, Hedvig Kjellström, and Stephan Mandt (2018). Advances in variational inference. IEEE transactions on pattern analysis and machine intelligence 41(8), 2008–2026.

A

Deferred results and proofs Chapter 2

A.1 Optimal coordinate update under the Mean-field approximating family

We will now prove Proposition 2. That is, we show for the Mean-field approximating family that the optimal variational density for component ℓ is

$$q_\ell^*(y_\ell) \propto \exp\left(\mathbb{E}_{-\ell}[\log p(Y, \mathcal{D})]\right).$$

In this problem we are maximising the ELBO. Let us now rewrite this objective as a function of q_ℓ, absorbing all others terms into a constant:

$$\begin{aligned}\mathcal{L}(q, \mathcal{D}) &= \mathbb{E}[\log p(Y, \mathcal{D})] - \mathbb{E}[\log q(Y)] \\ &\propto \mathbb{E}_\ell\left[\mathbb{E}_{-\ell}[\log p(Y, \mathcal{D})]\right] - \mathbb{E}_\ell[\log q_\ell(Y_\ell)], \end{aligned} \tag{A.1.1}$$

where we made use of the tower rule for the first term and the Mean-field assumption for the second term. Now, note that (A.1.1) can be rewritten as a negative KL divergence

$$\begin{aligned}\mathbb{E}_\ell\left[\mathbb{E}_{-\ell}[\log p(Y, \mathcal{D})]\right] - \mathbb{E}_\ell[\log q_\ell(Y_\ell)] &= \mathbb{E}_\ell\left[\log \exp\left(\mathbb{E}_{-\ell}[\log p(Y, \mathcal{D})]\right)\right] - \mathbb{E}_\ell[\log q_\ell(Y_\ell)] \\ &= -\mathbb{E}_\ell\left[\log\left(\frac{q_\ell(Y_\ell)}{\exp\left(\mathbb{E}_{-\ell}[\log p(Y, \mathcal{D})]\right)}\right)\right].\end{aligned}$$

It now trivially follows that we can maximise this by choosing the function q_ℓ that minimises the KL divergence which is $q_\ell^*(y_\ell) \propto \exp\left(\mathbb{E}_{-\ell}[\log p(Y, \mathcal{D})]\right)$.

A.2 CAVI updates for the exponential family

The idea of CAVI is that it iterates through all factors $\ell = 1, \ldots, d$, updates them using (2.2.5) and then repeats this cycle until convergence towards a (local) optimum is attained. The exact procedure of CAVI can be found in Algorithm 2.

Algorithm 2: Coordinate Ascent Variational Inference (CAVI)

Input: $(q_\ell)_{1\leqslant \ell \leqslant L}$: initial variational factors.
Output: Return the optimised Mean-field variational density q satisfying: for all $y \in \mathsf{Y}, q(y) = \prod_{\ell=1}^{d} q_\ell(y_\ell)$.
while *the ELBO has not converged* **do**
  for $\ell = 1, \dots, d$ **do**
    set $q_\ell(y_\ell) \propto \exp\left(\mathbb{E}_{-\ell}[\log p(Y, \mathcal{D})]\right)$
  end
  Compute the ELBO.
end

Moreover, when the complete conditionals $p(Y_\ell | Y_{-\ell}, \mathcal{D})$ belong to the exponential family, the coordinate update in (2.2.5) is available in closed form. This provides tractable updates for models such as Latent Dirichlet allocation (Blei et al., 2003) and Gaussian Mixture Models (Bishop, 2006). Let us now give an expression for these updates in the exponential family.

For the full definition of the exponential family, we refer to Definition 3. Applying this definition to the full conditionals of p gives that they are given by

$$p(y_\ell | y_{-\ell}, \mathcal{D}) = h(y_\ell) \exp\left(\langle v(y_{-\ell}, \mathcal{D}), S(y)\rangle_E - A(v(y_{-\ell}, \mathcal{D}))\right), \quad y \in \mathsf{Y},$$

and v maps the parameters $y_{-\ell}$ and $\mathcal{D}$ to a subset of $\operatorname{Int} E_0$. Now, using the CAVI update from Proposition 2, we find that

$$\begin{aligned} q_\ell^*(y_\ell) &\propto \exp\left(\mathbb{E}_{-\ell}[\log p(Y, \mathcal{D})]\right) \\ &\propto \exp\left(\mathbb{E}_{-\ell}[\log p(Y_\ell | Y_{-\ell}, \mathcal{D})]\right) \\ &= \exp\left(\log h(Y_\ell) + \langle \mathbb{E}_{-\ell}[v(Y_{-\ell}, \mathcal{D})], S(y)\rangle_E - E_{-\ell}[A(v(Y_{-\ell}, \mathcal{D}))]\right) \\ &\propto h(Y_\ell) \exp\left(\langle \mathbb{E}_{-\ell}[v(Y_{-\ell}, \mathcal{D})], S(y)\rangle_E\right), \end{aligned}$$

which demonstrates that the variational factors are in the same exponential family as the complete conditionals. The natural parameter of this variational factor of component ℓ is consequently given by

$$\zeta_\ell = \mathbb{E}_{-\ell}[v(Y_{-\ell}, \mathcal{D})].$$

A.3 Special cases in the α-divergence family (non-exhaustive)

A list of such special cases can be found in Table A.1. Moreover, notice that f_α is a convex function on $(0, \infty)$. From this and its form in (2.2.7), it can be seen that the α-divergence itself is a special case of the f-divergence (Morimoto, 1963; Csiszár, 1964).

α	Definition	Notes	References
$\alpha \to 0$	$\mathbb{D}_{\mathrm{KL}}(\mathbb{P} \parallel \mathbb{Q})$	Inclusive KL	Kullback and Leibler (1951)
$\alpha = 0.5$	$4\mathrm{Hel}^2(\mathbb{P} \parallel \mathbb{Q})$	Function of the square Hellinger distance	Hellinger (1909); Lindsay (1994)
$\alpha \to 1$	$\mathbb{D}_{\mathrm{KL}}(\mathbb{Q} \parallel \mathbb{P})$	Exclusive KL	Kullback and Leibler (1951)
$\alpha = 2$	$\frac{1}{2}\mathbb{D}_{\chi^2}(\mathbb{Q} \parallel \mathbb{P})$	Function of the χ^2-divergence	Dieng et al. (2017)

Table A.1: Special cases in the α-divergence family (non-exhaustive).

A.4 Illustration of the role of α using an unnormalised variational density

Recall that the function f_α in (2.2.8) in the definition of the α-divergence, also includes the $u-1$ terms. These disappear if we restrict q to integrate to 1. However, as also done by Minka et al. (2005), we can also visualise the role of α without making this requirement. The result is plotted in Figure A.1. For this figure, we also optimised the normalisation constant belonging to q.

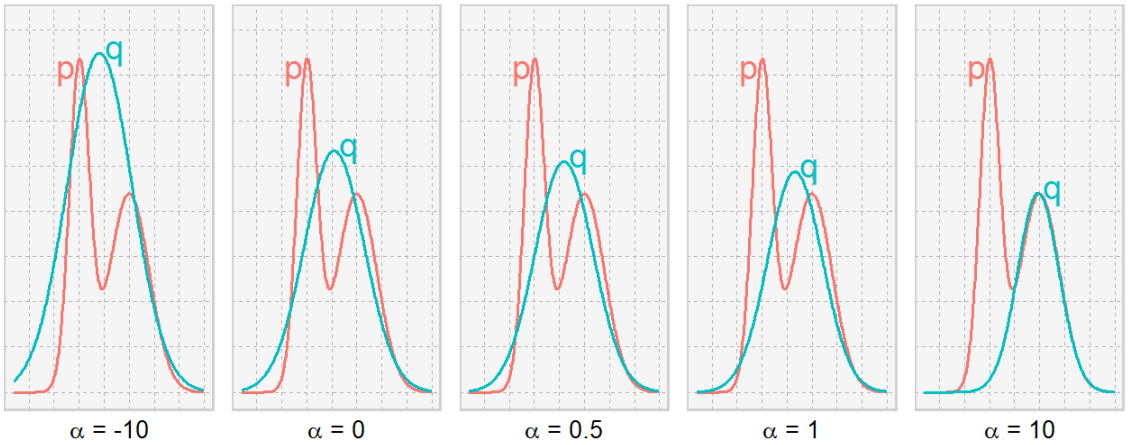


Figure A.1: Mixture of two Gaussians p and optimal (unnormalised) Gaussian q that minimises the α-divergence using different values of α. Figure was inspired by Minka et al. (2005), further details can be found in Appendix D.1.

A.5 Proof Theorem 1

Proof. We note that condition (2.3.4) is equivalent to

$$\int_{\mathsf{Y}} \varphi_n^{(\alpha)}(y) \log\left(\frac{k(\theta_{n+1}, y)}{k(\theta_n, y)}\right) \nu(\mathrm{d}y) \geqslant 0 \tag{A.5.1}$$

since $\alpha \in [0, 1)$. Moreover, plugging θ_n into the objective in (2.3.5) yields a value of zero. Therefore, choosing the argmax θ_{n+1} guarantees that the objective function (2.3.5)

is non-negative. Next to that, by Proposition 1 and the fact that $(b_n)_{n\geqslant 1}$ is non-negative, we also find for the second term in the integral

$$\int_{\mathsf{Y}} b_n k\left(\theta_n, y\right) \log\left(\frac{k(\theta, y)}{k\left(\theta_n, y\right)}\right) \nu(\mathrm{d}y) = -b_n \mathbb{D}_{\mathrm{KL}}(K(\theta_n, \cdot) \parallel K(\theta_{n+1}, \cdot)) \leqslant 0.$$

We conclude that (A.5.1) is satisfied as

$$\begin{aligned}\int_{\mathsf{Y}} \varphi_n^{(\alpha)}(y) \log\left(\frac{k(\theta_{n+1}, y)}{k(\theta_n, y)}\right) \nu(\mathrm{d}y) &\geqslant \int_{\mathsf{Y}} \left[\varphi_n^{(\alpha)}(y) + b_n k\left(\theta_n, y\right)\right] \log\left(\frac{k(\theta, y)}{k\left(\theta_n, y\right)}\right) \nu(\mathrm{d}y) \\ &\geqslant 0.\end{aligned}$$

□

A.6 Details updating equations given by (2.3.7)

Proof. In Corollary 6 of Daudel et al. (2022), it is shown that under a Gaussian setting, the argmax problem from Theorem 1 has a unique solution $\theta^* = (m^*, \Sigma^*)$ given by:

$$\begin{aligned} m^* &= \mathbb{E}[Y] \\ \Sigma^* &= \mathbb{C}\mathrm{ov}(Y). \end{aligned}$$

where Y is a d-dimensional random variable with density:

$$\psi(y) = \frac{1}{1+b}\check{\varphi}(y) + \frac{b}{1+b}\mathcal{N}(y; m_n, \Sigma_n) = \gamma\check{\varphi}(y) + (1-\gamma)\mathcal{N}(y; m_n, \Sigma_n).$$

The mean update trivially follows from this

$$\begin{aligned} m^* &= \int_{\mathsf{Y}} y\psi(y)\mathrm{d}y \\ &= \gamma \int_{\mathsf{Y}} y\check{\varphi}(y)\mathrm{d}y + (1-\gamma)\int_{\mathsf{Y}} y\mathcal{N}(y; m_n, \Sigma_n)\mathrm{d}y \\ &= \gamma\widehat{m}_n + (1-\gamma)m_n \end{aligned}$$

using the definition from (2.3.8). Next, we derive the update for the covariance which requires slightly more work:

$$\begin{aligned} \Sigma^* &= \int_{\mathsf{Y}} yy^\top\psi(y)\mathrm{d}y - m^*m^{*T} \\ &= \gamma\int_{\mathsf{Y}} yy^\top\check{\varphi}(y)\mathrm{d}y + (1-\gamma)\int_{\mathsf{Y}} yy^\top\mathcal{N}(y; m_n, \Sigma_n)\mathrm{d}y - m^*m^{*T} \\ &= \gamma(\widehat{\Sigma}_n + \widehat{m}_n\widehat{m}_n^\top) + (1-\gamma)(\Sigma_n + m_nm_n^T) - (\gamma\widehat{m}_n + (1-\gamma)m_n)(\gamma\widehat{m}_n + (1-\gamma)m_n)^T \\ &= \gamma\widehat{\Sigma}_n + (1-\gamma)\,\Sigma_n + \gamma\,(1-\gamma)\,(\widehat{m}_n - m_n)\,(\widehat{m}_n - m_n)^T. \end{aligned}$$

□

A.7 Details gradient-based approach for the exponential family

In their paper, Daudel et al. (2022) give an expression for the gradient of $g_n(\theta)$ (2.3.11) where the density $k(\theta, \cdot)$ belongs to the exponential family. Moreover, a condition for the smoothness-index β_n is also provided.

For the definition of the exponential family and the employed notation, we refer to the start of Section 3.2. Using this notation, we first consider the canonical case. Let $\zeta \in \operatorname{Int} E_0$ and set

$$g^{(o)}(\zeta) = -\int_{\mathsf{Y}} \check{\varphi}(y) \log\left(\frac{k^{(0)}(\zeta, y)}{k^{(0)}(\zeta_0, y)}\right) \nu(\mathrm{d}y), \quad \zeta \in \operatorname{Int} E_0.$$

For this function, the paper demonstrates (Proposition 5) that under some regularity conditions, for any convex subset $C_0 \subseteq \operatorname{Int} E_0$, the function $g^{(o)}$ is β_0-smooth over C_0 with

$$\nabla g^{(o)}(\zeta) = \nabla A(\zeta) - \int_{\mathsf{Y}} S\check{\varphi}\mathrm{d}\nu \quad \text{and} \quad \beta_0 \geqslant \sup_{\zeta \in C_0} \left\|\nabla\nabla^T A(\zeta)\right\|_{\mathrm{op}}, \tag{A.7.1}$$

where $\|\cdot\|_{\mathrm{op}}$ denotes the operator norm. Moreover, the paper demonstrates how this result can easily be generalised to a non-canonical case. In this case, we also multiply the auxiliary function by $(1-\alpha)^{-1}\int \varphi_n^{(\alpha)}\mathrm{d}\nu$ to mimic function g_n from Theorem 2. For this objective

$$g_n(\theta) = \int_{\mathsf{Y}} \frac{\varphi_n^{(\alpha)}(y)}{\alpha - 1} \log\left(\frac{k(\theta, y)}{k\left(\theta_n, y\right)}\right) \nu(\mathrm{d}y),$$

we find the following gradient

$$\nabla g_n(\theta) = \frac{\int \varphi_n^{(\alpha)}\mathrm{d}\nu}{1-\alpha} \nabla v\left(\theta\right) \cdot \left(\nabla A \circ v\left(\theta\right) - \int_{\mathsf{Y}} S\,\check{\varphi}\mathrm{d}\nu\right), \tag{A.7.2}$$

where v maps the parameter space T to a subset of $\operatorname{Int} E_0$. This result can now be applied in an Gaussian setting.

Consider the Gaussian case outlined in 2.3.1. Furthermore, in this section we further simplify this example by looking at $k(\theta, y) = \mathcal{N}(y; m, \Sigma)$ where the covariance matrix is now known and fixed $\Sigma = \sigma^2 I_d$ with $\sigma^2 > 0$ and therefore $\theta = m$. This belongs to exponential family as we can write $v(m) = \Sigma^{-1}m$ for the canonical kernel $k^{(o)}$ with $h(y) = (2\pi)^{-d/2}|\Sigma|^{-1/2}\mathrm{e}^{-y^T\Sigma^{-1}y/2}$, $S(y) = y$ and $A(\zeta) = \zeta^T\Sigma\zeta/2$. Furthermore, we can now apply (A.7.2) which yields the following update

$$\begin{aligned} m_{n+1} &= m_n - \frac{\gamma_n}{\beta_n}\frac{\int \varphi_n^{(\alpha)}\mathrm{d}\nu}{1-\alpha} \nabla v\left(m_n\right) \cdot \left(\nabla A \circ v\left(m_n\right) - \int_{\mathsf{Y}} S\,\check{\varphi}\mathrm{d}\nu\right) \\ &= m_n - \frac{\gamma_n}{\beta_n}\frac{\int \varphi_n^{(\alpha)}\mathrm{d}\nu}{(1-\alpha)} \Sigma^{-1}\left(m_n - \int_{\mathsf{Y}} y\,\check{\varphi}(y)\mathrm{d}y\right). \end{aligned}$$

This result is utilised for (2.3.12). Besides, it easily be seen that $\nabla\nabla^T g_n(\theta) = (1-\alpha)^{-1}\int \varphi_n^{(\alpha)}\mathrm{d}\nu\,\Sigma^{-1}$ from which the β-smoothness with $\beta = \sigma^{-2}(1-\alpha)^{-1}\int \varphi_n^{(\alpha)}\mathrm{d}\nu$ follows.

A.8 Generalised maximisation approach and gradient-based approach

The generalised result for the maximisation approach is given below.

Theorem 9 (Daudel et al. (2022)). *(Generalised maximisation approach). Assume (A1). Let $\alpha \in [0,1)$, let $(\eta_n)_{n\geqslant 1}$ be valued in $(0,1]$ and let $(\kappa_n)_{n\geqslant 1}$ be such that $(\alpha-1)\kappa_n \geqslant 0$ at all times $n \geqslant 1$. Furthermore, let $(b_{j,n})_{n\geqslant 1}$ be a non-negative sequence for all $j = 1 \dots J$. Starting from an initial parameter set $(\boldsymbol{\lambda}_1, \Theta_1) \in \mathcal{S}_J^+ \times \mathsf{T}^J$, let $(\boldsymbol{\lambda}_n, \Theta_n)_{n\geqslant 1}$ be defined iteratively for all $n \geqslant 1$ in such a way that (23) holds and*

$$\theta_{j,n+1} = \underset{\theta\in\mathsf{T}}{\operatorname{argmax}} \int_{\mathsf{Y}} \left[\varphi_{j,n}^{(\alpha)}(y) + b_{j,n} k\left(\theta_{j,n}, y\right)\right] \log\left(\frac{k(\theta, y)}{k\left(\theta_{j,n}, y\right)}\right) \nu(\mathrm{d}y), \quad j = 1 \dots J,$$

where we assume that this argmax is uniquely defined at each step. Then, $\Psi_\alpha(\mu_{n+1}k) \leqslant \Psi_\alpha(\mu_n k)$ for all $n \geqslant 1$.

Hence, the maximisation approach algorithm for the mixture case now follows from applying the updates in Theorems 3 and 9 for each $j = 1, \dots, J$. Thereby, it provides an algorithm that guarantees a systematic decrease in the mixture case. Alternatively, for the mixture component parameters optimisation, we can also apply the generalised gradient-based approach as shown in the theorem below.

Theorem 10 (Daudel et al. (2022)). *(Generalised gradient-based approach). Assume (A1). Let $\mathsf{T} \subseteq \mathbb{R}^d$ be a convex set, let $\alpha \in [0,1)$, let $(\eta_n)_{n\geqslant 1}$ be valued in $(0,1]$ and let $(\kappa_n)_{n\geqslant 1}$ be such that $(\alpha-1)\kappa_n \geqslant 0$ at all times n. Furthermore, for all $j = 1 \dots J$, let $(\gamma_{j,n})_{n\geqslant 1}$ be valued in $(0,1]$. Starting from an initial parameter set $(\boldsymbol{\lambda}_1, \Theta_1) \in \mathcal{S}_J^+ \times \mathsf{T}^J$, let $(\boldsymbol{\lambda}_n, \Theta_n)_{n\geqslant 1}$ be defined iteratively for all $n \geqslant 1$ in such a way that (23) holds and*

$$\theta_{j,n+1} = \theta_{j,n} - \frac{\gamma_{j,n}}{\beta_{j,n}} \nabla g_{j,n}\left(\theta_{j,n}\right), \quad j = 1 \dots J,$$

where for all $j = 1 \dots J$, $(g_{j,n})_{n\geqslant 1}$ is defined by: for all $n \geqslant 1$ and all $\theta \in \mathsf{T}$,

$$g_{j,n}(\theta) = \int_{\mathsf{Y}} \frac{\varphi_{j,n}^{(\alpha)}(y)}{\alpha - 1} \log\left(\frac{k(\theta, y)}{k\left(\theta_{j,n}, y\right)}\right) \nu(\mathrm{d}y)$$

and $g_{j,n}$ is assumed to be $\beta_{j,n}$-smooth. Then, $\Psi_\alpha(\mu_{n+1}k) \leqslant \Psi_\alpha(\mu_n k)$ for all $n \geqslant 1$.

Thus, by applying the updates in Theorems 3 and 10 for each $j = 1, \dots, J$, an alternative updating scheme is provided that guarantees a systematic decrease in Ψ_α while simultaneously updating the mixture weights and mixture component parameters.

B

Deferred results and proofs Chapter 3

B.1 Lemma 1

Lemma 1. *Let*

$$a_n = \gamma_n(1-\alpha)\frac{\widehat{\sigma}_n^2}{\sigma^2} = \gamma_n \frac{(1-\alpha)\sigma_n^2}{\alpha\sigma^2 + (1-\alpha)\sigma_n^2}.$$

If the sequence $(\gamma_n)_{n\geqslant 1}$ has a minimum $\gamma_{\min} > 0$, then the sequence $(a_n)_{n\geqslant 1}$ is bounded from below by a strictly positive constant.

Proof. We do already have a lower bound on γ_n for all $n \geqslant 1$ given by $\gamma_{\min}$. Let us now inspect the remaining term given by

$$\frac{(1-\alpha)\sigma_n^2}{\alpha\sigma^2 + (1-\alpha)\sigma_n^2}.$$

Note that this term strictly lies between 0 and 1. If we find a lower bound σ_n^2, we also obtain a lower bound for this fraction. Recall that the covariances are updated using

$$\sigma_{n+1}^2 = \gamma_n \widehat{\sigma}_n^2 + (1-\gamma_n)\,\sigma_n^2 + \gamma_n\,(1-\gamma_n)\,(\widehat{m}_n - m_n)^2\,, \tag{B.1.1}$$

where all terms are clearly strictly positive. Now, crucially, the update for the "proposal" covariance $\widehat{\sigma}_n^2$, does not depend on the mean updates and is given by

$$\widehat{\sigma}_n^2 = \frac{\sigma_n^2\sigma^2}{\alpha\sigma^2 + (1-\alpha)\sigma_n^2}.$$

As also derived in the proof of Theorem 4, it satisfies

$$\widehat{\sigma}_n^2 = \frac{\sigma_1^2\sigma^2}{\sigma_1^2 + \alpha^n(\sigma^2 - \sigma_1^2)}.$$

It is easy to see that this sequence $(\widehat{\sigma}_n^2)_{n\geqslant 1}$ converges monotonically to σ^2. It therefore follows that $\widehat{\sigma}_n^2 \geqslant \min\{\sigma_1^2, \sigma^2\}$ for all $n \geqslant 1$. By (B.1.1), this implies a lower bound on σ_n^2 given by $\sigma_n^2 \geqslant \gamma_{\min} \min\{\sigma_1^2, \sigma^2\}$. We now conclude that the sequence $(a_n)_{n\geqslant 1}$ is lower bounded by

$$a_{\min} = \gamma_{\min}^2 \frac{(1-\alpha)\min\{\sigma_1^2, \sigma^2\}}{\alpha\sigma^2 + (1-\alpha)\gamma_{\min}\min\{\sigma_1^2, \sigma^2\}}.$$

□

B.2 Lemma 2

Lemma 2. *Using the updating equations from Theorem 4,*

$$\widehat{\sigma}_n^2 = \frac{\sigma_n^2\sigma^2}{\alpha\sigma^2 + (1-\alpha)\sigma_n^2} \quad \text{and} \quad \sigma_{n+1}^2 = \widehat{\sigma}_n^2$$

where $\alpha \in [0, 1)$*, it holds that*

$$\lim_{n\to\infty} \frac{\left(\frac{\widehat{\sigma}_{n+1}^2}{\sigma_{n+1}^2}\right)^n}{\left(\frac{\widehat{\sigma}_n^2}{\sigma_n^2}\right)^{n-1}} = 1.$$

Proof. Let us introduce the following shorthand notation

$$x_n = \frac{\widehat{\sigma}_n^2}{\sigma_n^2} \quad \text{and} \quad y_n = \widehat{\sigma}_n^2.$$

It trivially follows that $x_n = y_n/y_{n-1}$, $x_n \to 1$, and $y_n \to \sigma^2$. Using this notation, we now want to prove the following

$$\lim_{n\to\infty} \frac{x_{n+1}^n}{x_n^{n-1}} = 1.$$

As $x_n \to 1$, we may also look at $\frac{x_{n+1}^n}{x_n^{n-1}}\frac{1}{x_n} = \left(\frac{x_{n+1}}{x_n}\right)^n = \left(\frac{y_{n+1}y_{n-1}}{y_n^2}\right)^n$. The term inside the brackets can be rewritten to

$$\begin{aligned}\frac{y_{n+1}y_{n-1}}{y_n^2} &= \frac{y_{n-1}}{y_n}\frac{\sigma^2}{\alpha\sigma^2 + (1-\alpha)y_n} \\ &= \frac{\alpha\sigma^2 + (1-\alpha)y_{n-1}}{\alpha\sigma^2 + (1-\alpha)y_n} \\ &= 1 + (1-\alpha)\frac{y_{n-1} - y_n}{\alpha\sigma^2 + (1-\alpha)y_n}.\end{aligned}$$

Hence, we now aim to establish the convergence of

$$\lim_{n\to\infty} \left(1 + (1-\alpha)\frac{y_{n-1} - y_n}{\alpha\sigma^2 + (1-\alpha)y_n}\right)^n = 1. \tag{B.2.1}$$

Note that by the Binomial Theorem, for some z, it holds $(1+z)^n = 1 + nz + o(nz)$ where nz satisfies $nz \to 0$. This means that if $z = o(1/n)$, then $(1+z)^n = 1 + o(1)$ which

converges to 1. Therefore, we know that it is sufficient for us to prove that $y_{n-1} - y_n = o(1/n)$, i.e., $\lim_{n\to\infty} n(y_{n-1} - y_n) = 0$. Let us now rewrite this in terms of $(y_{n-1} - \sigma^2)$

$$\begin{aligned} y_{n-1} - y_n &= y_{n-1} - \frac{\sigma^2 y_{n-1}}{\alpha\sigma^2 + (1-\alpha)y_{n-1}} \\ &= \frac{\alpha\sigma^2 y_{n-1} + (1-\alpha)y_{n-1}^2 - \sigma^2 y_{n-1}}{\alpha\sigma^2 + (1-\alpha)y_{n-1}} \\ &= (1-\alpha)\frac{y_{n-1}}{\alpha\sigma^2 + (1-\alpha)y_{n-1}}(y_{n-1} - \sigma^2). \end{aligned} \tag{B.2.2}$$

Now, note that in the limit the fraction in (B.2.2) converges to 1. Recall that in the proof of Theorem 4, we already demonstrated that $y_n = \widehat{\sigma}_n^2 \to \sigma^2$ at exponential speed. From this we infer that $y_{n-1} - y_n \to 0$ at exponential speed. This is sufficient for $y_{n-1} - y_n = o(1/n)$ to hold. Hence, it follows that (B.2.1) is true which concludes the proof. □

B.3 Proof Theorem 5

Before we prove Theorem 5, we will give a useful lemma. For this lemma, we recall the definition of *Loewner order*. For two symmetric matrices, we write $A \succ B$ when $A - B$ is positive definite and we say that A is strictly larger in *Loewner order*. Moreover, we write $A \succeq B$ when $A - B$ is positive semi-definite. For further details, we refer the reader to Bhatia (2013). Next to that, in the following section $\boldsymbol{I_d}$ denotes the $d \times d$ identity matrix. Let us now present the following lemma.

Lemma 3. *Let A and B be symmetric, positive definite matrices and let $C = \alpha A + (1-\alpha)B$ where $\alpha \in [0, 1)$. If A and B are commutable, then all eigenvalues of $\alpha C^{-1}A$ lie strictly between 0 and 1.*

Proof. It immediately follows that C is also symmetric and positive definite and so is its inverse. Therefore, $\alpha C^{-1}A$ is a product of two positive definite matrices. By Meenakshi and Rajian (1999), we know that $\alpha C^{-1}A \succeq \mathbf{O}$ iff it is also symmetric. Now note that:

$$(\alpha C^{-1}A)^T = \alpha C^{-1}A \iff AC^{-1} = C^{-1}A \iff AC = CA \iff AB = BA.$$

Hence, we get that $\alpha C^{-1}A \succeq \mathbf{O}$ iff A and B are commutable. Moreover, we get that:

$$\boldsymbol{I_d} - \alpha C^{-1}A = (1-\alpha)C^{-1}B \succeq \mathbf{O}$$

by a similar argument. Hence, $\mathbf{O} \preceq \alpha C^{-1}A \preceq \boldsymbol{I_d}$. Moreover, since the product of invertible matrices is also invertible, we know that $\alpha C^{-1}A \succ \mathbf{O}$ and $(1-\alpha)C^{-1}B \succ \mathbf{O}$. It then follows that $\mathbf{O} \prec \alpha C^{-1}A \prec \boldsymbol{I_d}$. By the min-max theorem (Lieb and Loss, 2001) and the fact that $0 < x^T \alpha C^{-1}Ax < 1 \; \forall x \in \mathbb{R}^n$, we find that all eigenvalues lie between 0 and 1. □

We are now ready to prove Theorem 5.

Theorem 5

Proof. We know that $p(y) = c \times \mathcal{N}(y; m, \Sigma)$ and $k(\theta_n, y) = \mathcal{N}(y; m_n, \Sigma_n)$. Then, by completing the square we obtain that:

$$\begin{aligned}
\breve{\varphi}_n^{(\alpha)}(y) &\propto k(\theta_n, y)^\alpha p(y)^{1-\alpha}, \\
&\propto \exp\left\{-\frac{1}{2}\alpha(y-m_n)^T\Sigma_n^{-1}(y-m_n) - \frac{1}{2}(1-\alpha)(y-m)^T\Sigma^{-1}(y-m)\right\}, \\
&\propto \exp\left\{-\frac{1}{2}y^T(\alpha\Sigma_n^{-1} + (1-\alpha)\Sigma^{-1})y + y^T(\alpha\Sigma_n^{-1}m_n + (1-\alpha)\Sigma^{-1}m)\right\}, \\
&\propto \mathcal{N}(y; \widehat{m}_n, \widehat{\Sigma}_n),
\end{aligned}$$

where

$$\begin{aligned}
\widehat{m}_n &= \widehat{\Sigma}_n\left(\alpha\Sigma_n^{-1}m_n + (1-\alpha)\Sigma^{-1}m\right), \\
\widehat{\Sigma}_n &= \left(\alpha\Sigma_n^{-1} + (1-\alpha)\Sigma^{-1}\right)^{-1}.
\end{aligned} \tag{B.3.1}$$

The updating equations (2.3.7) for the mean can then be written as:

$$\begin{aligned}
m_{n+1} &= \gamma_n\widehat{\Sigma}_n\left(\alpha\Sigma_n^{-1}m_n + (1-\alpha)\Sigma^{-1}m\right) + (1-\gamma_n)m_n, \\
&= \underbrace{\gamma_n(1-\alpha)\widehat{\Sigma}_n\Sigma^{-1}}_{A_n}m + \underbrace{\left(\boldsymbol{I_d} - \gamma_n\left(\boldsymbol{I_d} - \alpha\widehat{\Sigma}_n\Sigma_n^{-1}\right)\right)}_{B_n}m_n.
\end{aligned}$$

Now, note that $(1-\alpha)\widehat{\Sigma}_n\Sigma^{-1}$ is of the form of matrix C in Lemma 3 (since diagonal matrices are commutable). Using this we can, similarly as in Lemma 1, find a diagonal constant matrix $A_{\min}$ that satisfies $\boldsymbol{O} \prec A_{\min} \prec A_n$ for all $n \geqslant 1$. As this matrix is diagonal, this implies that the eigenvalues of $A_{\min}$ are strictly between 0 and 1. In the same vein, we can define $\tilde{m}_{n+1} = \tilde{m}_n + A_{\min}(m - \tilde{m}_n)$ for all $n \geqslant 1$ with $\tilde{m}_1 = m_1$. Then, it can be shown by induction that

$$\begin{aligned}
\tilde{m}_n &= (\boldsymbol{I_d} - A_{\min})^{n-1}m_1 + \sum_{i=0}^{n-2}(\boldsymbol{I_d} - A_{\min})^i A_{\min}m \\
&= (\boldsymbol{I_d} - A_{\min})^{n-1}m_1 + \left(\boldsymbol{I_d} - (\boldsymbol{I_d} - A_{\min})^{n-1}\right)(A_{\min})^{-1}A_{\min}m \\
&= m + (\boldsymbol{I_d} - A_{\min})^{n-1}(m_1 - m)
\end{aligned}$$

where we used Proposition 1.5.38 of Hubbard and Hubbard (2015) in the second step. Now, by t Consequently, we obtain that

$$\lim_{n\to\infty}\tilde{m}_n - m = \lim_{n\to\infty}A_{\min}^{n-1}(m_1 - m) = \lim_{n\to\infty}\left(PD^{n-1}P^T\right)(m_1 - m) = \mathbf{0},$$

where PDP^{-1} denotes the spectral decomposition of $A_{\min}$. By the Sandwich Theorem and the fact that $A_{\min} \prec A_n$ for all $n \geqslant 1$, it now follows that $\lim_{n\to\infty} m_n = m$. Since, m_n is a convex combination of $\widehat{m}_n$ and its previous value, this also proves that $\lim_{n\to\infty}\widehat{m}_n = m$. Therefore, the optimal regularisation sequence is given by $\gamma_n \equiv 1$. Under this choice of γ_n, the convergence rate is proportional to $\alpha\widehat{\Sigma}_n\Sigma_n^{-1}$ where we again see the proportionality to α.

The proof for the convergence of Σ_n can easily be proven by writing the update equation (2.3.7) in terms of diagonal matrices and then noting that the element-wise update are the same as in the 1-dimensional case. The proof for the mean update could also simplified in this way but it was chosen to keep the general matrix notation as the above proof also holds for non-diagonal matrices which are still commutable. □

B.4 Definitions smoothness and strong convexity

In this section, $||\cdot||$ denotes the Euclidean norm.

For the following two definitions, we assume that $g : \mathbb{R}^d \mapsto \mathbb{R}$ is continuously differentiable. We say that g is β-*smooth* if for any $\theta, \theta' \in \mathbb{R}^d$ we have:

$$||\nabla g(\theta) - \nabla g(\theta')|| \leqslant \beta ||\theta - \theta'||.$$

Moreover, we say that g is smooth if there exists some $\beta > 0$ such that g is β-smooth.

In the literature (Vial, 1982), we commonly say that a function is α-strongly convex. To avoid confusion in this dissertation, we will always use $\tilde{\alpha} > 0$ to denote the strong convexity index and α when we refer to the α-divergence. We then say that g is $\tilde{\alpha}$-*strongly convex* if for any $\theta, \theta' \in \mathbb{R}^d$ we have

$$g(\theta) - g(\theta') \leqslant \nabla g(\theta)^\top (\theta - \theta') - \frac{\tilde{\alpha}}{2} ||\theta - \theta'||.$$

The duality between these two definitions becomes even clearer when we look at the definitions for the case where g is twice differentiable. That is, g is β-*smooth* if for any $\theta \in \mathbb{R}^d$ we have

$$\nabla \nabla^\top g(\theta) \preceq \beta \boldsymbol{I_d}$$

and g is $\tilde{\alpha}$-*smooth* if for any $\theta \in \mathbb{R}^d$ we have

$$\nabla \nabla^\top g(\theta) \succeq \tilde{\alpha} \boldsymbol{I_d}.$$

Hence, smoothness bounds the Hessian from above and strong convexity bounds the Hessian from below in Loewner order. This illustrates why we can derive strong convergence results under both of these assumptions. As we know some particular 'range' in which the function can change, we can adapt our step size based on that such we never make steps that are way too large or small. This is illustrated in Figure B.1.

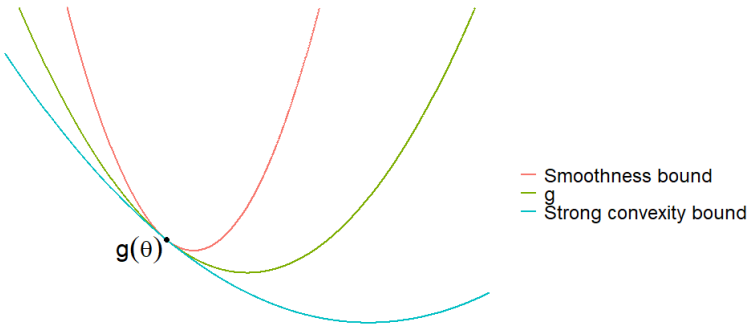


Figure B.1: Visualisation of a particular function g and a corresponding smoothness and strong convexity bound.

B.5 Proof Theorem 6

Before we prove Theorem 6, we first give two important lemmas. The first one of these can be also be found in appendix A.1 of Daudel et al. (2022) but is included for the sake of completeness as we need it for both Theorem 2 and 6.

Lemma 4. *Let $\gamma \in (0, 1]$ and let $g : \mathbb{R}^d \mapsto \mathbb{R}$ be a β-smooth function. Then, for all $x \in \mathbb{R}^d$ we find that*

$$g(\theta) - g\left(\theta - \frac{\gamma}{\beta}\nabla g(\theta)\right) \geqslant \frac{\gamma}{2\beta}\|\nabla g(\theta)\|^2.$$

Proof. A different characterization of β-smoothness under convexity is the following, we have that for all $\theta, \theta' \in \mathbb{R}^d$ (Bubeck et al., 2015)

$$g\left(\theta'\right) - g(\theta) \leqslant \nabla g(\theta)^\top(\theta' - \theta) + \frac{\beta}{2}\left\|\theta - \theta'\right\|^2.$$

If we now set $\theta' = \theta - \frac{\gamma}{\beta}\nabla g(\theta)$, we obtain that:

$$\begin{aligned} g\left(\theta - \frac{\gamma}{\beta}\nabla g(\theta)\right) - g(\theta') &\leqslant -\frac{\gamma}{\beta}\|\nabla g(\theta)\|^2 + \frac{\gamma^2}{2\beta}\|\nabla g(\theta)\|^2 \\ &\leqslant -\frac{\gamma}{\beta}\left(1 - \frac{\gamma}{2}\right)\|\nabla g(\theta)\|^2. \end{aligned}$$

Since $\gamma \in (0, 1]$, it now easily follows that

$$g(\theta) - g\left(\theta - \frac{\gamma}{\beta}\nabla g(\theta)\right) \geqslant \frac{\gamma}{2\beta}\|\nabla g(\theta)\|^2.$$

□

Lemma 5. *Let $g : \mathbb{R}^d \mapsto \mathbb{R}$ be an $\tilde{\alpha}$-strongly convex and β-smooth function. Then for any $\theta, \theta' \in \mathbb{R}^d$, we get that*

$$g\left(\theta - \frac{1}{\beta}\nabla g\left(\theta\right)\right) - g\left(\theta'\right) \leqslant -\frac{1}{2\beta}\left\|\nabla g(\theta)\right\|^2 + \nabla g(\theta)^\top(\theta - \theta') - \frac{\tilde{\alpha}}{2}\|\theta - \theta'\|^2,$$

Proof. Applying Lemma 4 under $\gamma = 1$ and the definition of $\tilde{\alpha}$-strong convexity yields

$$\begin{aligned} g\left(\theta - \frac{1}{\beta}\nabla g\left(\theta\right)\right) - g(\theta') &= g\left(\theta - \frac{1}{\beta}\nabla g\left(\theta\right)\right) - g(\theta) + g(\theta) - g(\theta') \\ &\leqslant -\frac{1}{2\beta}\left\|\nabla g(\theta)\right\|^2 + \nabla g(\theta)^\top(\theta - \theta') - \frac{\tilde{\alpha}}{2}\|\theta - \theta'\|^2. \end{aligned}$$

□

Theorem 6

Proof. This proof is based on the proof of Theorem 3.10 in the book by Bubeck et al. (2015). The only difference is that we prove the unconstrained case while the book proves the constrained case. Note that plugging $\theta' = \theta^*$ into Lemma 5 gives that

$$\frac{1}{2\beta}\left\|\nabla g(\theta)\right\|^2 - \nabla g(\theta)^\top(\theta - \theta') \leqslant -\frac{\tilde{\alpha}}{2}\|\theta - \theta'\|^2$$

as $g\left(\theta - \frac{1}{\beta}\nabla g\left(\theta\right)\right) - g(\theta^*)$ is non-negative. Using this inequality, we now find that

$$\begin{aligned}
\left\|\theta_{n+1} - \theta^*\right\|^2 &= \left\|\theta_n - \frac{1}{\beta}\nabla f\left(\theta_n\right) - \theta^*\right\|^2 \\
&= \left\|\theta_n - \theta^*\right\|^2 - \frac{2}{\beta}\nabla g\left(\theta_n\right)^\top\left(\theta_n - \theta^*\right) + \frac{1}{\beta^2}\left\|\nabla g\left(\theta_n\right)\right\|^2 \\
&\leqslant \left(1 - \frac{\tilde{\alpha}}{\beta}\right)\left\|\theta_n - \theta^*\right\|^2 \\
&\leqslant \left(1 - \frac{\tilde{\alpha}}{\beta}\right)^n\left\|\theta_1 - \theta^*\right\|^2
\end{aligned}$$

which concludes the proof.

□

C

Deferred results Chapter 4

C.1 Details Monte Carlo estimator of VR bound given by (2.3.17)

Let us first give an explicit expression for the Monte Carlo estimator used by Daudel et al. (2022) in their implementation of the GMM algorithm. In this algorithm, we draw independently M samples $(Y_{m,n})_{1\leq m\leq M}$ from proposal q_n. Moreover, note that the variational density, denoted by $\mu_n k$, only coincides with q_n for the IS-n approach. Then, the implemented importance sampling Monte Carlo estimator of the VR bound (suggested by Li and Turner (2016)) at time n is given by

$$
\begin{aligned}
\widehat{\mathcal{L}}_{\alpha,M}(q,\mathcal{D}) &= \frac{1}{1-\alpha}\log\frac{1}{M}\sum_{m=1}^{M}\frac{p\left(Y_{m,n}^{(j)}\right)^{1-\alpha}\left(\mu_n k\left(Y_{m,n}^{(j)}\right)\right)^{\alpha}}{q\left(Y_{m,n}^{(j)}\right)} \\
&= \frac{1}{1-\alpha}\log\frac{1}{M}\sum_{m=1}^{M}\frac{\mu_n k\left(Y_{m,n}^{(j)}\right)}{q\left(Y_{m,n}^{(j)}\right)}\left(\frac{\mu_n k\left(Y_{m,n}^{(j)}\right)}{p\left(Y_{m,n}^{(j)}\right)}\right)^{\alpha-1} \\
&= \frac{1}{1-\alpha}\log\frac{1}{M}\sum_{m=1}^{M}\sum_{j=1}^{J}\lambda_{j,n}\frac{k\left(Y_{m,n}^{(j)}\right)}{q\left(Y_{m,n}^{(j)}\right)}\left(\frac{\mu_n k\left(Y_{m,n}^{(j)}\right)}{p\left(Y_{m,n}^{(j)}\right)}\right)^{\alpha-1} \\
&= \frac{1}{1-\alpha}\log\frac{1}{M}\sum_{m=1}^{M}\sum_{j=1}^{J}\lambda_{j,n}\widehat{\varphi}_{j,n}^{(\alpha)}\left(Y_{m,n}^{(j)}\right) \\
&= \frac{1}{1-\alpha}\log\frac{1}{M}\sum_{j=1}^{J}\lambda_{j,n}\sum_{m=1}^{M}\widehat{\varphi}_{j,n}^{(\alpha)}\left(Y_{m,n}^{(j)}\right),
\end{aligned}
$$

where the final expression shows that we compute the VR bound without much additional cost as we already computed $\sum_{m=1}^{M} \widehat{\varphi}_{j,n}^{(\alpha)}\left(Y_{m,n}^{(j)}\right)$ for $j = 1, \ldots, J$ at time n (note that we need this term for both the mean and weights updates). Moreover, the experiment is replicated 30 times and we take a Monte Carlo estimator again, so in that sense we are using a 'double' Monte Carlo estimator.

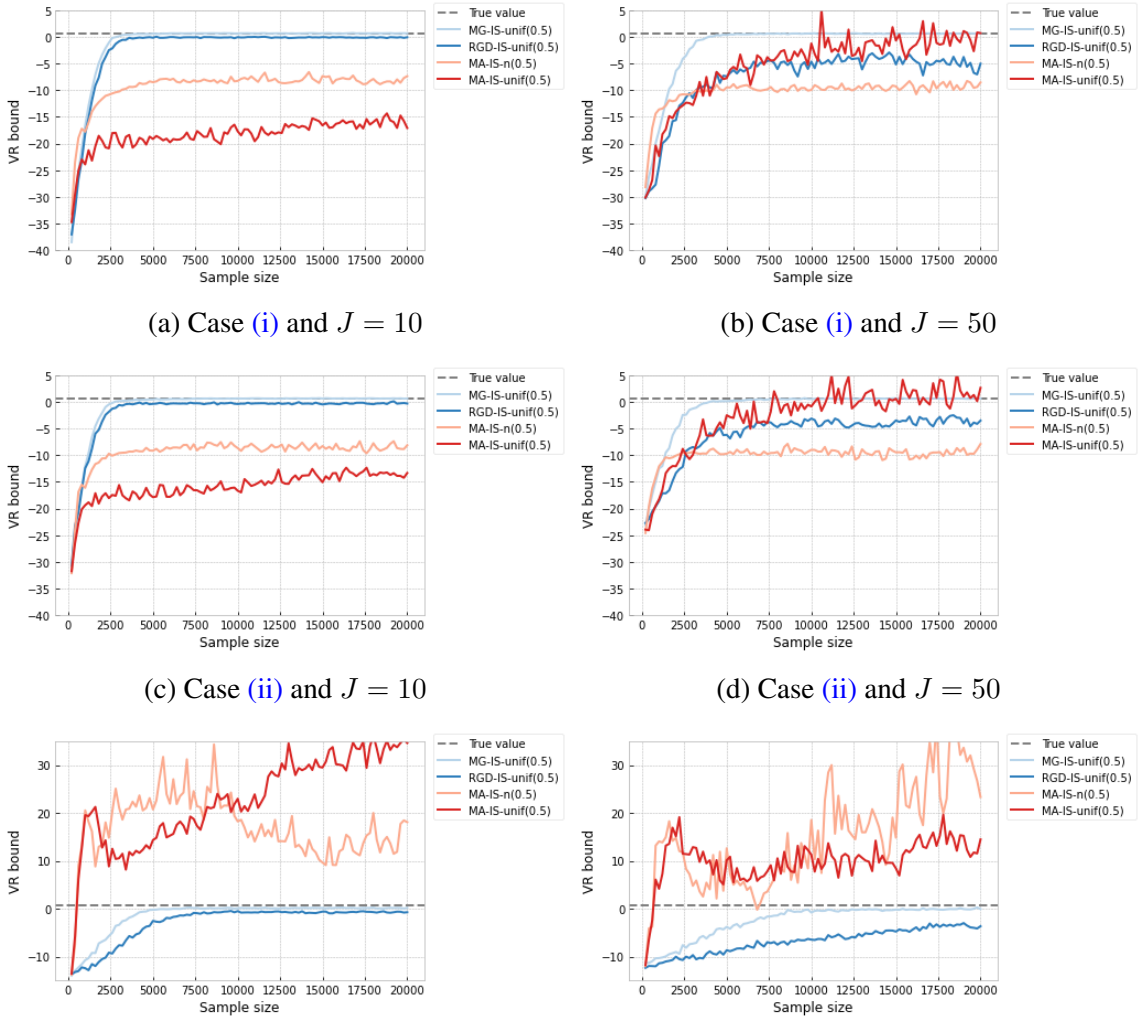
We compare this estimator with the true value of the VR bound. Using the notation that $q^* = p(\cdot|\mathcal{D})$, we know that

$$\begin{aligned}
\mathcal{L}_\alpha(q^*, \mathcal{D}) &= \frac{1}{1-\alpha} \log\left(\int_{\mathrm{Y}} q^*(y)^\alpha p(y, \mathcal{D})^{1-\alpha} \nu(\mathrm{d}y)\right) \\
&= \frac{1}{1-\alpha} \log\left(\int_{\mathrm{Y}} q^*(y)^\alpha \left(c \times q^*(y)\right)^{1-\alpha} \nu(\mathrm{d}y)\right) \\
&= \log(c),
\end{aligned}$$

which is used for the figures containing the VR bound in this dissertation.

C.2 Covariance updates for cases (i), (ii), and (iii) under $\gamma = 0.5$

We observe in Figure 4.2 that the VR bounds of the MA algorithm are highly volatile. Especially in case (iii), it highly overshoots the VR bound and therefore these results are not reliable. For cases (i) and (ii), we notice that the algorithm including covariance updates is outperformed by the initial algorithm. However, notably, the MA-IS-unif approach seems to work better when $J = 50$ compared to $J = 10$. This suggests that using a high number of mixture components is generally better for this approach.



(a) Case (i) and $J = 10$

(b) Case (i) and $J = 50$

(c) Case (ii) and $J = 10$

(d) Case (ii) and $J = 50$

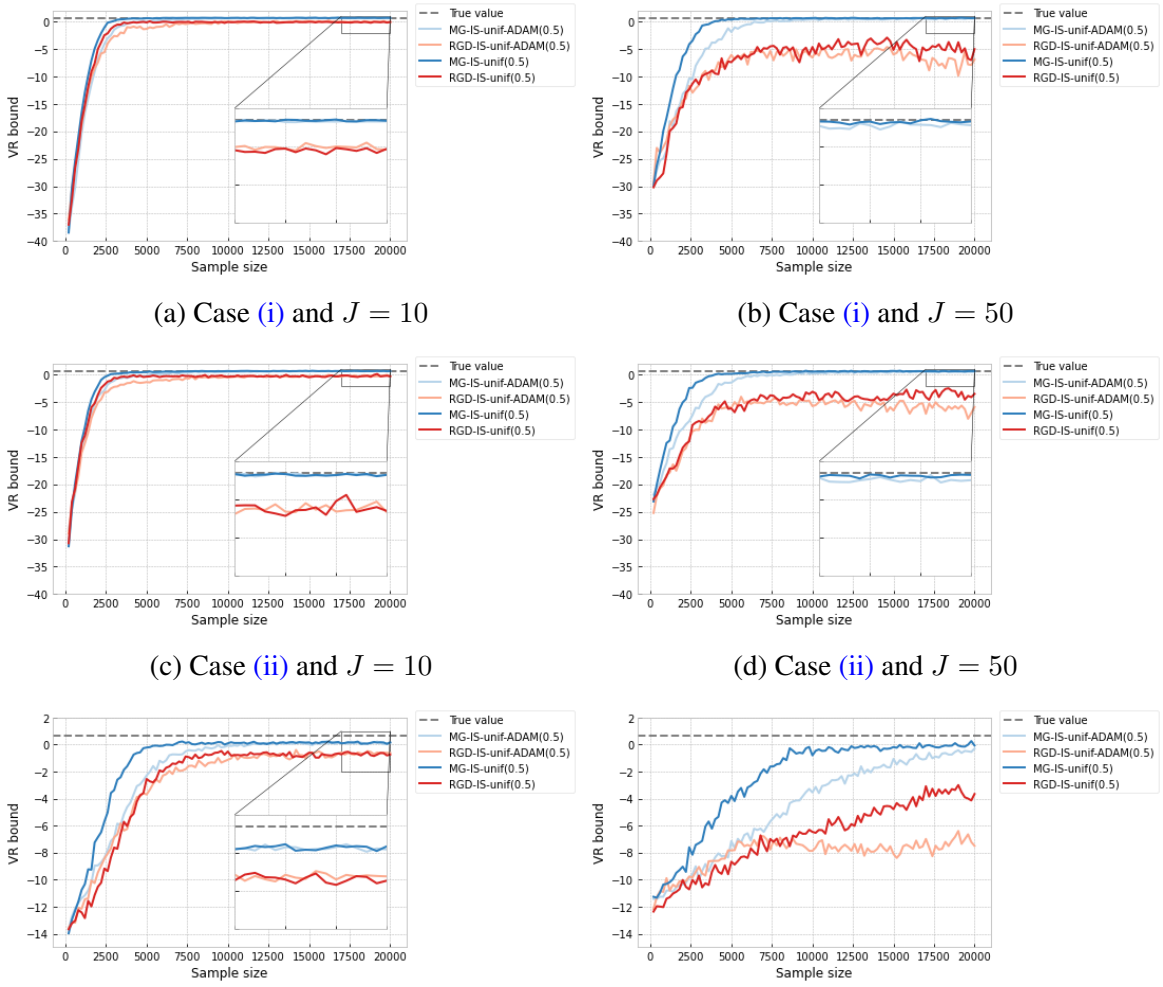
(e) Case (iii) and $J = 10$

(f) Case (iii) and $J = 50$

Figure C.1: Implementation of Algorithm 1 including the covariance updates (referred to as MA) compared with the MG and RGD approach using the IS-unif sampler and $\gamma = 0.5$. Moreover, $\alpha = 0.2$, $d = 16$, $M = 200$, $\kappa_n = 0$ and $\eta_n = 0.1$ for all n.

C.3 ADAM for the IS-unif sampler and $\gamma = 0.5$

In Figure C.2, the results of the implementation of ADAM for the IS-unif sampler can be found. We find that the existing algorithm outperforms the ADAM approach in all cases.



(a) Case (i) and $J = 10$

(b) Case (i) and $J = 50$

(c) Case (ii) and $J = 10$

(d) Case (ii) and $J = 50$

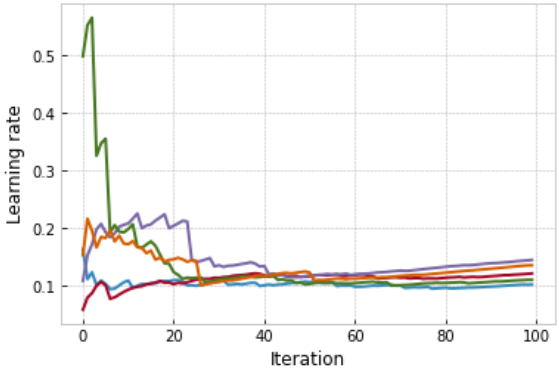
(e) Case (iii) and $J = 10$

(f) Case (iii) and $J = 50$

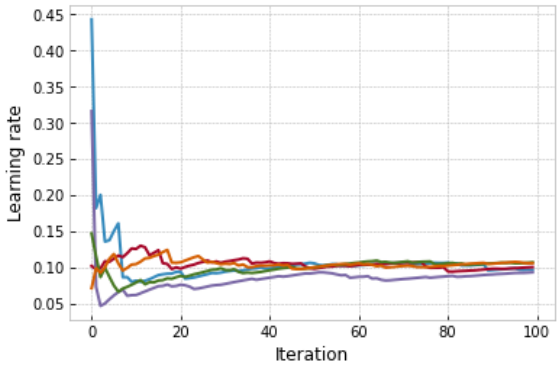
Figure C.2: Implementation of ADAM for the maximisation approach and RGD using the IS-unif sampler and $\gamma = 0.5$. Moreover, $\alpha = 0.2$, $d = 16$, $M = 200$, $\kappa_n = 0$ and $\eta_n = 0.1$ for all n.

C.4 Learning rate ADAM for the IS-unif sampler and case (iii)

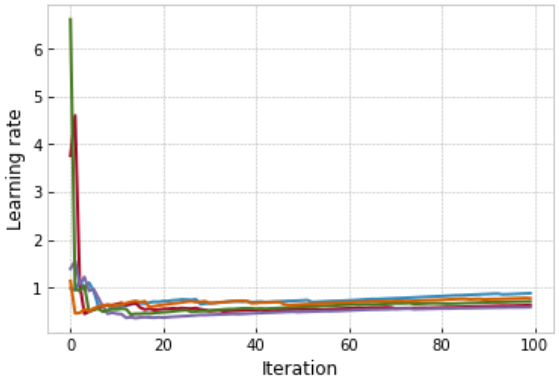
In Figure C.3, the learning rates of the first component of the MG-IS-unif-ADAM(γ) algorithm for 5 different replications are visualised. In contrast with Figure 4.6, here case (iii) was used instead of (i). Nonetheless, we observe similar patterns. At the start there are some large fluctuations with the learning rate occasionally getting larger than 1. After some time, the learning rate gets more stable and keeps increasing slightly.



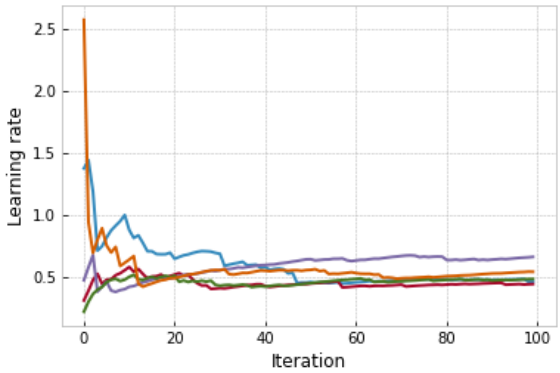
(a) Case (iii), $\gamma = 0.1$, and $J = 10$



(b) Case (iii), $\gamma = 0.1$, and $J = 50$



(c) Case (iii), $\gamma = 0.5$, and $J = 10$



(d) Case (iii), $\gamma = 0.5$, and $J = 50$

Figure C.3: Learning rate corresponding to the first component in the MG-IS-unif-ADAM(γ) approach for 5 different replications of the algorithm with $N = 100$ iterations. Moreover, $\alpha = 0.2$, $d = 16$, $M = 200$, $\kappa_n = 0$ and $\eta_n = 0.1$ for all n.

D

Code appendix

D.1 R code

Part of the coding, mainly for illustrative purposes, has been done in R. Below, we only show the code for Figures 2.1 and 2.2. For the remaining figures we refer to the following Github repository: www.github.com/StanKoobs/MSc-Dissertation-Variational-Inference.

D.1.1 Packages.R

```
#############################################################################
### In this script we define the packages to be used
#############################################################################

# List with the packages which we need
ListofPackages <-c("tidyverse", "mvtnorm", "LaplacesDemon", "cowplot",
                   "latex2exp", "tikzDevice")
NewPackages <- ListofPackages[!(ListofPackages %in%
                                  installed.packages()[,"Package"])]

# Install packages not on system so far
if (length(NewPackages) > 0) {
  install.packages(NewPackages)
}

# Import all packages
lapply(ListofPackages, require, character.only = TRUE)
```

```
# No need to keep these variables
rm(ListofPackages, NewPackages)
```

D.1.2 DissertationggTheme.R

```
#############################################################################
### In this script we define our own ggplot theme
#############################################################################

DissertationggTheme <- function() {
  theme(panel.background = element_rect(fill = "grey96"),
        axis.line = element_blank(),
        panel.border = element_rect(colour = "gray82", fill = NA,
                                    size = 1.5),
        panel.grid.major = element_line(colour = "grey70",
                                        linetype = "dashed"),
        panel.grid.minor = element_line(colour = "grey70",
                                        linetype = "dashed"),
        axis.text.x = element_text(size= 13),
        axis.text.y = element_text(size= 13),
        axis.title.x = element_text(size = 15),
        axis.title.y = element_text(size = 15),
        axis.ticks = element_line(size = 1.5),
        axis.ticks.length = unit(1.5, "mm")
  )
}
```

D.1.3 Figure 2.1

```
#############################################################################
### Replicating Figure 10.2 of Bishop (2006)
#############################################################################

source("Packages.R")
source("DissertationggTheme.R")

# Define mean and covariance matrix of p(y|D)
mu <- c(0, 0)
Sigma <- matrix(c(1, 0.97, 0.97, 1), ncol = 2, nrow = 2)

# Find its value on grid of points
x <- seq(-3, 3, by = 0.025)
npoints <- length(x)
data <- matrix(c(rep(x, npoints),
                 rep(x, each = npoints)), ncol = 2)
```

```r
# Define objective function to be minimised
KL <- function(vec, type = "Forward") {
  mu1 <- vec[1]; mu2 <- vec[2]; sigma1 <- vec[3]; sigma2 <- vec[4]
  densityzcx <- dmvnorm(data, mu, Sigma)
  densityz <- dmvnorm(data, c(mu1, mu2), diag(c(sigma1, sigma2)))
  result <- KLD(densityz, densityzcx)
  if (type == "Forward") {
    result$sum.KLD.px.py
  } else if (type == "Reverse") {
    result$sum.KLD.py.px
  }
}

# As this is an easy 2-dimensional problem, we do not need an efficient
# approach. Therefore, we use numerical KL minimisation by using the
# Nelder-Mead method (default of the optim() function)
par <- optim(c(0, 0, 0.3, 0.3), KL)$par
muvar <- par[1:2]
sigmavar <- diag(par[3:4])

# Target density
zcondx <- function(x, y) {
  dmvnorm(cbind(x, y), mean = mu, sigma = Sigma)
}
# Variational density
variational <- function(x, y) {
  dmvnorm(cbind(x, y), mean = muvar, sigma = sigmavar)
}


findlevelCurves <- function(mu, sigma, func = zcondx) {
  q1 <- qmvnorm(0.6827, mean = mu, sigma = sigma)$quantile
  q2 <- qmvnorm(0.9545, mean = mu, sigma = sigma)$quantile
  q3 <- qmvnorm(0.9973, mean = mu, sigma = sigma)$quantile

  if (func == "zcondx") {
    c(zcondx(q1, q1), zcondx(q2, q2), zcondx(q3, q3))
  } else if (func == "variational") {
    c(variational(q1, q1), variational(q2, q2), variational(q3, q3))
  }
}

# Values of the corresponding 1, 2, and 3-sigma contours
breaksp <- findlevelCurves(mu, Sigma, "zcondx")
breaksq <- findlevelCurves(muvar, sigmavar, "variational")
```

```
# Include all data in one data frame
data <- as_tibble(data)
names(data) <- c("x", "y")
datap <- data %>% mutate(z = zcondx(x, y), c = as.factor("Conditional"))
dataq <- data %>% mutate(z = variational(x, y),
                         c = as.factor("Variational"))

ForwardPlot <- ggplot() +
  stat_contour(data = datap,
               aes(x = x, y = y, z = z, colour = "$p(y|\\mathcal{D})$"),
               breaks = breaksp, size = 1) +
  stat_contour(data = dataq,
               aes(x = x, y = y, z = z, colour = "$q(y)$"),
               breaks = breaksq, size = 1) +
  labs(color = "Distribution") +
  DissertationggTheme() +
  ggtitle("Exclusive KL") +
  theme(legend.title = element_text(size = 13),
        legend.text = element_text(size = 13),
        plot.title = element_text(size = 18, face = "bold"),
        axis.title.x = element_blank(),
        axis.title.y = element_blank()) +
  xlim(-4.1, 4.1) +
  ylim(-4.1, 4.1)
  #scale_x_continuous(breaks = c(-2.5, 0, 2.5)) +
  #scale_y_continuous(breaks = c(-2.5, 0, 2.5))

ForwardPlot

options(tz = "Europe/Berlin")

tikz(file = "KLDivForward.tex", width = 5, height = 4)

ForwardPlot

dev.off()

# In the part below, we repeat the work done above for the reverse KL
# divergence
x <- seq(-4.25, 4.25, by = 0.025)
npoints <- length(x)
data <- matrix(c(rep(x, npoints),
                 rep(x, each = npoints)), ncol = 2)

par <- optim(c(0, 0, 1, 1), KL, type = "Reverse")$par
```

```
muvar <- par[1:2]
sigmavar <- diag(par[3:4])

breaksp <- findlevelCurves(mu, Sigma, "zcondx")
breaksq <- findlevelCurves(muvar, sigmavar, "variational")

data <- as_tibble(data)
names(data) <- c("x", "y")
datap <- data %>% mutate(z = zcondx(x, y))
dataq <- data %>% mutate(z = variational(x, y))

ReversePlot <- ggplot() +
  stat_contour(data = datap,
               aes(x = x, y = y, z = z, colour = "$p(y|\\mathcal{D})$"),
               breaks = breaksp, size = 1) +
  stat_contour(data = dataq,
               aes(x = x, y = y, z = z, colour = "$q(y)$"),
               breaks = breaksq, size = 1) +
  labs(color = "Distribution") +
  DissertationggTheme() +
  ggtitle("Inclusive KL") +
  theme(legend.title = element_text(size = 13),
        legend.text = element_text(size = 13),
        plot.title = element_text(size = 18, face = "bold"),
        axis.title.x = element_blank(),
        axis.title.y = element_blank())
ReversePlot

options(tz = "Europe/Berlin")

tikz(file = "KLDivForward.tex", width = 5, height = 4)

ReversePlot

dev.off()

# Combine both plots in one figure
plot_grid(ForwardPlot, ReversePlot)
```

D.1.4 Figure 2.2

```
#########################################################################
### Replicating Figure 1 of Minka (2005) without normalisation
#########################################################################
```

```
source("Packages.R")
source("DissertationggTheme.R")

# Point grid
x <- seq(-1.8, 2, by = 0.01)

# Mixture of Gaussians we will be trying to approximate
p <- function(y) {
  0.45 * dnorm(y, -0.5, 0.2) + 0.55 * dnorm(y, 0.5, 0.4)
}

# Function for alpha divergence for alpha is not zero or one
falpha <- function(u, alpha) {
  1 / (alpha * (alpha - 1)) * (u^alpha - 1)
}

# Alpha-divergence function
alphadiv <- function(py, qy, alpha) {
  # qy and py are vectors of probability densities expected to be of the
  # same length

  if (length(py) != length(qy)) {
    stop("py and qy not of equal length")
  }

  if (alpha == 0) {
    #KLD(py, qy)$sum.KLD.py.px
    sum(qy * (log(qy) - log(py)))
  } else if (alpha == 1) {
    #KLD(py, qy)$sum.KLD.px.py
    sum(py * (log(py) - log(qy)))
  } else {
    #sum(falpha(qy / py, alpha = alpha) * py)
    1 / (alpha * (1 - alpha)) *
      sum(qy - py^alpha * qy^(1 - alpha))
  }
}

# Now we use q(y) = N(y; mu, sigma) as our variational family
# This leads to the following objective function
objective <- function(vec, alpha) {
  # vec is expected to be 2-dim input vector with mu and sigma
  mu <- vec[1]; sigma <- vec[2]
  densityp <- p(x)
  densityq <- dnorm(x, mu, sigma)
  alphadiv(densityq, densityp, alpha = alpha)
```

```
}

# As it is a simple 1D problem, we use a simple built-in optimisation
# algorithm. That is, we use the  method "L-BFGS-B" of Byrd et. al.
# (1995) which allows box constraints, that is, each variable can be
# given a lower and/or upper bound.
findParameters <- function(alpha) {
  optim(c(0.4, 0.35),
        objective,
        alpha = alpha,
        lower = c(-Inf, 0),
        upper = c(2, 2),
        method = "L-BFGS-B")$par
}

# Generate sequence of py values
py <- p(x)

# Function to generate desired plots for particular alpha
generateplot <- function(alpha) {
  par <- findParameters(alpha)

  qy <- dnorm(x, par[1], par[2])

  # Used for plotting "q" at the right spot in the figure
  xcoord <- x[which.max(qy)]
  ycoord <- max(qy)

  ggplot() +
    geom_line(aes(x = x, y = py, colour = "py"), size = 1.2) +
    geom_line(aes(x = x, y = qy, colour = "qy"), size = 1.2) +
    theme(plot.title = element_text(hjust = 0.5, size = 16,
                                    face = "bold")) +
    ThesisggTheme() +
    theme(legend.position = "none",
          axis.text.x = element_blank(),
          axis.ticks.x = element_blank(),
          axis.title.x = element_text(size = 20),
          axis.text.y = element_blank(),
          axis.ticks.y = element_blank(),
          axis.title.y = element_blank()) +
    annotate(geom = "text", x = -0.75, y = 0.9, label = "p",
             color = "#F8766D", size = 10) +
    annotate(geom = "text", x = xcoord + 0.3, y = ycoord + 0.02,
             label = "q", color = "#00BFC4", size = 10) +
    xlab(TeX(paste0("$\\alpha$ = ", as.character(alpha))))
```

```
}

# Generate the five figures
pm10 <- generateplot(-10)
p0 <- generateplot(0)
p05 <- generateplot(0.5)
p1 <- generateplot(1)
p10 <- generateplot(10)

# Final figure
plot_grid(pm10, p0, p05, p1, p10, ncol = 5)
```

D.2 Python code

The code framework was kindly provided by Dr. Kamélia Daudel. In the code below, this framework is slightly adjusted and extended in some places. We only show a few main functions to highlight how the code works. The rest will be uploaded to Github once the code corresponding to Daudel et al. (2022) has also been publicly released.

D.2.1 Class to run the algorithm

The class ALGO_TEMPLATE is the main class for the algorithm and all algorithms are derived from this.

```
import numpy as np
from scipy.stats import multivariate_normal
from sklearn.metrics import mean_squared_error


def sample_from_k_theta_1d(mean, sd, nb_samples):
    return np.random.normal(mean, sd, nb_samples)


class ALGO_TEMPLATE:

    def __init__(self, i, lnprob, D, thetas_init, alpha, N, J, \
                 sigma_n, M_n, M_eval, eta_n, kappa_n, gamma_n,\
                     bool_unif, adam):
        '''
        Initialise parameters
        :param i: used for paralellisation (int)
        :param lnprob: function which computes the log prob of a
            given model (function)
        :param D: dimension of the latent space (int)
        :param thetas_init: initial set (theta_1, ..., theta_J)
        :param alpha: parameter of the alpha-divergence (float)
        :param N: number of iterations (int)
```

```
        :param J: number of components (int)
        :param sigma_n: bandwidth (float > 0)
        :param M_n: number of samples used to estimate bmu (int)
        :param eta_n: learning rate (float > 0)
        :param bool_unif: if True, use a uniform sampler
                          if False, use the best sampler at time
                            n
        :param kappa_n: constant appearing in the mixture
            weights update
        :param adam: if 0, use standard learning
                          if 1, use momentum learning
                          if 2, use ADAM learning
        '''
        # Model
        self.lnprob = lnprob
        self.D = D

        # Initial thetas
        self.thetas_init = thetas_init.copy()

        # Parameters
        self.alpha = alpha
        self.N = N
        self.J = J
        self.sigma_n = sigma_n
        self.M_n = M_n
        self.M_eval = M_eval
        self.eta_n0 = eta_n
        self.eta_n = eta_n
        self.gamma_n = gamma_n
        self.kappa_n = kappa_n
        self.adam = adam

        if self.D == 1:
            self._generate_from_multivariate = \
                sample_from_k_theta_1d
        else:
            self._generate_from_multivariate =  \
                self._sample_from_k_theta_nd

        if bool_unif:
            self.compute_repartition = \
                self._compute_repartition_unif
            self.sampler = self._sampler_unif
        else:
            self.compute_repartition = \
            self._compute_repartition_best_n
            self.sampler = self._sampler_best_n

```

```
    # Some helper functions
    def _sample_from_k_theta_nd(self, mean, sd, nb_samples):
        return np.random.multivariate_normal(mean=mean, \
            cov=sd * np.identity(self.D), size=nb_samples)

    def _compute_repartition_unif(self, weights):
        uniform_weights = np.array([1 / self.J] * self.J)
        return np.random.multinomial(self.M_n, uniform_weights)

    def _compute_repartition_best_n(self, weights):
        return np.random.multinomial(self.M_n, weights)

    def _sampler_unif(self, k_theta_y, muk_y):
        return 1 / self.J * sum(k_theta_y)

    def _sampler_best_n(self, k_theta_y, muk_y):
        return muk_y

    def _compute_repartition_eval(self, weights):
        return np.random.multinomial(self.M_eval, weights)

    def _estimation(self, weights, thetas):
        samples = []
        repartition = self._compute_repartition_eval(weights)
        for j in range(self.J):
            nb = repartition[j]
            for _ in range(nb):
                y = self._generate_from_multivariate(thetas[j],\
                    self.sigma_n, nb)[0]
                samples.append(y)

        mean_estimation = np.mean(samples, axis=0)
        covar_estimation = np.cov(np.array(samples).T)

        return mean_estimation, covar_estimation

    # Core funtions
    def _updates_n(self, thetas, weights):
        # to be implemented for each subclass
        return 0

    def _updates_n_adam(self, thetas, firstmom, secondmom, \
        weights):
        # to be implemented for each subclass
        return 0

    def _transition_n(self, thetas, weights_init):
        self.eta_n = self.eta_n0

```

```
        weights, means, llh_n, renyi_bound_n = \
            self._updates_n(thetas, weights_init.copy())

        # Additional computations to assess convergence
        mean_estimation, covar_estimation = \
            self._estimation(weights, means)

        return weights, means, [llh_n], [renyi_bound_n], \
            [mean_estimation], covar_estimation

    def _transition_n_adam(self, thetas, weights_init, \
        firstmom, secondmom, n):
            self.eta_n = self.eta_n0

        weights, means, llh_n, renyi_bound_n, firstmom,\
            secondmom, learning_rate = \
            self._updates_n_adam(thetas, weights_init.copy(),\
                firstmom, secondmom, n)

        # Additional computations to assess convergence
        mean_estimation, covar_estimation = \
            self._estimation(weights, means)

        return weights, means, [llh_n], [renyi_bound_n], \
            [mean_estimation], covar_estimation, firstmom, \
            secondmom, learning_rate

    # Maximisation Gradient Monte Carlo algorithm

    def _algo(self):

        thetas = self.thetas_init.copy()

        firstmom = np.zeros(self.D)
        secondmom = np.zeros(self.D)

        n = 1

        # Init with uniform mixture weights
        if self.adam == 0:
            weights, thetas, llh, renyi_bound, mean_estimation,\
                covar_estimation = self._transition_n(thetas,  \
                    np.array([1 / self.J] * self.J))
        else:
            weights, thetas, llh, renyi_bound, mean_estimation,\
                covar_estimation, firstmom, secondmom, \
                learning_rate = self._transition_n_adam(thetas,\
                    np.array([1 / self.J] * self.J), \
                    firstmom, secondmom, n)
```

```
                learning_save = np.array([learning_rate])

        renyi_bound_lst = renyi_bound
        llh_lst = llh
        mean_estimation_lst = mean_estimation

        weights_save = weights
        thetas_save = thetas
        covar_estimation_save = covar_estimation

        # When using momentum (or adam) we use a different
        # transitioning function that also updates the first
        # (and second moment)
        if self.adam == 0:
            while n < self.N:
                print(n)

                n += 1

                weights, thetas, llh, renyi_bound, \
                mean_estimation, covar_estimation, \
                learning_rate = \
                    self._transition_n(thetas, weights)

                renyi_bound_lst.extend(renyi_bound)
                llh_lst.extend(llh)
                mean_estimation_lst.extend(mean_estimation)

                covar_estimation_save = covar_estimation
                thetas_save = thetas
        else:
            while n < self.N:
                print(n)

                # We already increase n as we already used
                # transition_n above using uniform weights
                n += 1

                weights, thetas, llh, renyi_bound,\
                mean_estimation, covar_estimation, firstmom,\
                secondmom, learning_rate =\
                    self._transition_n_adam(thetas, weights,\
                        firstmom, secondmom, n)

                renyi_bound_lst.extend(renyi_bound)
                llh_lst.extend(llh)
                mean_estimation_lst.extend(mean_estimation)

                learning_save = learning_rate[0]
```

```
                covar_estimation_save = covar_estimation
                thetas_save = thetas

        return thetas_save, weights_save, renyi_bound_lst,\
            llh_lst, mean_estimation, covar_estimation_save,\
            learning_save
```

D.2.2 Subclass for the MG_MC algorithm

One example of a subclass is MG_MC which can be used for the MG-IS-unif(γ) and MG-IS-n(γ) approaches and their enhanced versions including momentum and ADAM.

```
import numpy as np
from scipy.stats import multivariate_normal
from sklearn.metrics import mean_squared_error
import algo_template

def sample_from_k_theta_1d(mean, sd, nb_samples):
    return np.random.normal(mean, sd, nb_samples)


class MG_MC(algo_template.ALGO_TEMPLATE):

    def __init__(self, i, lnprob, D, thetas_init, alpha, N, J, \
        sigma_n, M_n, M_eval, eta_n, kappa_n, gamma_n,\
        bool_unif, bool_adam):
        super().__init__(i, lnprob, D, thetas_init, alpha, N,\
            J, sigma_n, M_n, M_eval, eta_n, kappa_n, gamma_n, \
            bool_unif, bool_adam)

        if self.alpha == 0.:
            self.compute_objective =
                self._compute_objective_zero
        else:
            self.compute_objective = \
                self._compute_renyi

    def _compute_objective_zero(self, weights, weights_inter, \
        alpha_zero_inter):
        return 1/self.M_n * alpha_zero_inter

    def _compute_renyi(self, weights, weights_inter,\
        alpha_zero_inter):
        renyi_bound_inter = np.sum(weights * weights_inter)
        return 1 / (1 - self.alpha) * np.log(1 / self.M_n *\
            renyi_bound_inter)

    def _updates_n(self, thetas, weights):
```

```
37
38          # Init intermediate quantities
39          weights_inter = np.zeros(self.J)
40          means_inter = np.zeros((self.J, self.D))
41          means_inter_sum = np.zeros(self.J)
42          llh_inter = 0
43          alpha_zero_inter = 0
44          renormalise = 0
45
46          repartition = self.compute_repartition(weights)
47
48          for j in range(self.J):
49              nb = repartition[j]
50              for _ in range(nb):
51              # Compute intermediate quantities appearing in phi
52                  y = self._generate_from_multivariate(thetas[j],\
53                      self.sigma_n, 1)[0]
54                  k_theta_y = multivariate_normal.pdf(thetas, \
55                      mean = y, \
56                      cov = self.sigma_n * np.identity(self.D))
57                  muk_y = sum([weights[i] * k_theta_y[i] \
58                      for i in range(self.J)])
59                  lnprob = self.lnprob(y)
60                  # Sample from proposal q
61                  q_y = self.sampler(k_theta_y, muk_y)
62                  log_ratio = np.log(muk_y) - lnprob
63
64                  # Compute phi
65                  phi = 1 / q_y * np.exp((self.alpha - 1) * \
66                      log_ratio) * k_theta_y
67
68                  # Update intermediate quantities
69                  weights_inter += phi
70                  means_inter += \
71                      [phi[l] * y for l in range(self.J)]
72                  means_inter_sum += phi
73
74                  # Additional computations to assess the
75                  # convergence
76                  lnprob_multiplied = np.exp(lnprob) / q_y
77                  llh_inter += lnprob_multiplied
78                  alpha_zero_inter += lnprob_multiplied * \
79                      (-log_ratio)
80
81          # Compute parameters updates
82          weights_update = weights * np.power(weights_inter +
83              self.kappa_n, self.eta_n)
84          weights_update_sum = np.sum(weights_update)
85          weights_update_normalised = weights_update /
```

```
            weights_update_sum

        means_inter_normalised = means_inter /
            means_inter_sum.reshape(-1, 1)
        # reshape(-1, 1) makes the numpy array a vector
        means_update = np.reshape((1. - self.gamma_n) * thetas +
            self.gamma_n * means_inter_normalised, (-1, self.D))

        # Update convergence estimators
        renyi_bound = self.compute_objective(weights,
            weights_inter, alpha_zero_inter)
        llh = np.log(1 / self.M_n * llh_inter)

        #self.container_weights.append(weights)
        #self.container_thetas.append(thetas)

        return weights_update_normalised, means_update, llh,
            renyi_bound

    def _updates_n_adam(self, thetas, weights, firstmom,
        secondmom, n):

            beta1 = 0.9
            beta2 = 0.999
            epsilon = 1e-8

            # Init intermediate quantities
            weights_inter = np.zeros(self.J)
            means_inter = np.zeros((self.J, self.D))
            means_inter_sum = np.zeros(self.J)
            llh_inter = 0
            alpha_zero_inter = 0
            renormalise = 0

            repartition = self.compute_repartition(weights)


            for j in range(self.J):
                nb = repartition[j]
                for _ in range(nb):
                # Compute intermediate quantities appearing in
                # phi
                    y = self._generate_from_multivariate(
                        thetas[j], self.sigma_n, 1)[0]
                    k_theta_y = multivariate_normal.pdf(thetas,
                    mean = y,
                    cov = self.sigma_n * np.identity(self.D))
                    muk_y = sum([weights[i] * k_theta_y[i]
                        for i in range(self.J)])
```

```
                    lnprob = self.lnprob(y)
                    # Sample from proposal q
                    q_y = self.sampler(k_theta_y, muk_y)
                    log_ratio = np.log(muk_y) - lnprob

                    # Compute phi
                    phi = 1 / q_y * np.exp((self.alpha - 1) *
                        log_ratio) * k_theta_y

                    # Update intermediate quantities
                    weights_inter += phi
                    means_inter += [phi[l] * y
                        for l in range(self.J)]
                    means_inter_sum += phi

                    # Additional computations to assess the
                    # convergence
                    lnprob_multiplied = np.exp(lnprob) / q_y
                    llh_inter += lnprob_multiplied
                    alpha_zero_inter += lnprob_multiplied *
                        (-log_ratio)

            # Compute parameters updates
            weights_update = weights * np.power(weights_inter +
                self.kappa_n, self.eta_n)
            weights_update_sum = np.sum(weights_update)
            weights_update_normalised = weights_update /
                weights_update_sum

            means_inter_normalised = means_inter /
                means_inter_sum.reshape(-1, 1)
            # reshape(-1, 1) makes the numpy array a vector
            firstmom = beta1 * firstmom + (1 - beta1) *
                np.reshape(thetas - means_inter_normalised,
                    (-1, self.D))
            secondmom = beta2 * secondmom + (1 - beta2) *
                np.reshape((thetas - means_inter_normalised) **
                    2, (-1, self.D))

            firstmomhat = firstmom / (1 - beta1**n)
            secondmomhat = secondmom / (1 - beta2**n)

            if self.adam == 1:
                means_update = np.reshape(thetas -
                    self.gamma_n * firstmomhat, (-1, self.D))
            else:
                means_update = np.reshape(thetas -
                    self.gamma_n * firstmomhat /
                        (np.sqrt(secondmomhat) + epsilon),
```

```
                        (-1, self.D))
            learning_rate = np.reshape(self.gamma_n /
                (np.sqrt(secondmomhat) + epsilon), (-1, self.D))

            # Update convergence estimators
            renyi_bound = self.compute_objective(weights,
                weights_inter, alpha_zero_inter)
            llh = np.log(1 / self.M_n * llh_inter)

            return weights_update_normalised, means_update, llh,
                renyi_bound, firstmom, secondmom, learning_rate
```

D.2.3 Main function to launch experiments

We have only included the experiments to launch the MG algorithm with ADAM for the sake of conciseness. Other approaches are launched in the same way.

```
import numpy as np
import sys
from joblib import Parallel, delayed
from pathlib import Path

# Import algorithms
import mg_mc
import rgd_mc

# Import models
import toy_models

# Main function to launch the algorithms

def main_function(j):
    # Initialise (theta_1, ..., theta_J_t) according to a normal
    distribution
    if dim_latent == 1:
        thetas_init = np.random.normal(q0_mean, q0_sd, J)
    else:
        thetas_init = np.random.multivariate_normal(q0_mean,
            q0_sd * np.identity(dim_latent), J)

    #'''
    try:

        MG_MC_unif_adam = mg_mc.MG_MC(0, model.lnprob,
            dim_latent, thetas_init, alpha, N, J, sigma_n, M_n,
            M_eval, eta_n, kappa_n, gamma_n, True, 2)
        thetas_final, weights_final, renyi_bound_lst, llh_lst,
            mean_estimation, covar_estimation, learning_save =
                MG_MC_unif_adam._algo()
```

```
32
33          algo_str = "MG_MC_adam"
34          save_all_files(j, directory, algo_str, str_params,
35              renyi_bound_lst, llh_lst, mean_estimation,
36              covar_estimation, weights_final, thetas_final,
37              name_model, learning_save)
38      except:
39          pass
40      #'''
41      #'''
42      try:
43          MG_MC_n_adam = mg_mc.MG_MC(0, model.lnprob, dim_latent,
44              thetas_init, alpha, N, J, sigma_n, M_n, M_eval,
45              eta_n, kappa_n, gamma_n, False, 2)
46          thetas_final, weights_final, renyi_bound_lst, llh_lst,
47          mean_estimation, covar_estimation, learning_save =
48              MG_MC_n_adam._algo()
49
50          algo_str = "MG_MC_n_adam"
51          save_all_files(j, directory, algo_str, str_params,
52              renyi_bound_lst, llh_lst, mean_estimation,
53              covar_estimation, weights_final, thetas_final,
54              name_model, learning_save)
55      except:
56          pass
57      #'''
58      return 0
59
60  #name_model = "mixture"
61  #name_model = "imbalanced_mixture"
62  #name_model = "stud_mixture"
63
64  nmlst = ["mixture", "imbalanced_mixture", "stud_mixture"]
65
66  for name_model in nmlst:
67      ##############################################################
68      # Initialise parameters
69      ##############################################################
70      tot = 20000
71      nb_cores_used = 15
72      nb_repeat_exp = 30
73      alpha = 0.2
74      sigma_n = 1.
75      dim_latent = 16
76      kappa_n = 0.
77      M_n = 200
78      eta_n = 0.1
79      gamma_n_list = [0.1, 0.5, 1.0]
80      J_list = [10, 50]
```

```
    M_n_list = [20, 100, 200]

    for gamma_n in gamma_n_list:
        for J in J_list:
            for M_n in M_n_list:
                N = int(tot / M_n)
                M_eval = M_n
                str_params = '_M' + str(M_n) + 'N' + str(N) + \
                    'sigma' + str(sigma_n) + "J" + str(J) + "_"

                # Define the initial sampler q0
                q0_sd = 10.
                q0_mean = np.array([0.] * dim_latent)

                # Choose the targeted density and initialise
                # model
                models = {
                    "mixture": 1,
                    "imbalanced_mixture": 2,
                    "stud_mixture": 3,
                }

                if models[name_model] == 1:
                    target_means = [[2.] * dim_latent, [-2.] *
                        dim_latent]
                    target_nb_peaks = len(target_means)
                    target_sd = 1.
                    target_weights = [1. / target_nb_peaks] *
                        target_nb_peaks
                    Z = 2.

                    model = toy_models.GMM(target_nb_peaks,
                        target_means, target_weights, target_sd,
                        Z, dim_latent)

                if models[name_model] == 2:
                    target_means = [[2.] * dim_latent, [-2.] *
                        dim_latent, [1.] * dim_latent]
                    target_nb_peaks = len(target_means)
                    target_sd = 1.
                    target_weights = [0.25, 0.35, 0.4]
                    Z = 2.

                    model = toy_models.GMM(target_nb_peaks,
                        target_means, target_weights, target_sd,
                        Z, dim_latent)
                if models[name_model] == 3:
                    target_means = [[2.] * dim_latent, [-2.] *
                        dim_latent]
```

```
                    target_nb_peaks = len(target_means)
                    target_sd = 1.
                    target_weights = [1. / target_nb_peaks] *
                        target_nb_peaks
                    Z = 2.

                    model = toy_models.SMM(target_nb_peaks,
                        target_means, target_weights, target_sd,
                        Z, dim_latent)

                nb_samples_true = 1000
                samples_true =
                    model.sample_from_true_posterior(
                        nb_samples_true)
                mean_true_estimation = np.mean(samples_true,
                    axis = 0)
                covar_true_estimation =
                    np.cov(np.array(samples_true).T)

                # Set directory to save the target parameters
                directory = './results/dim' + str(dim_latent) +
                    '/'
                Path(directory).mkdir(parents = True,
                    exist_ok = True)
                save_file(0, directory, name_model +
                    "_MGRGD_target_mean_with_alpha" +
                    str(alpha) + str_params,
                    mean_true_estimation)
                save_file(0, directory, name_model +
                    "_MGRGD_target_covar_with_alpha"+ str(alpha)
                    + str_params, covar_true_estimation)

                # Set directory to save the outputs of the
                # experiments
                directory = './results/dim' + str(dim_latent) +
                    "/alpha" + str(alpha) + "/eta" + str(eta_n)\
                        + 'gamma' + str(gamma_n) + 'kappa' + \
                        str(kappa_n) + 'J' + str(J) + '/'
                Path(directory).mkdir(parents = True,
                    exist_ok = True)

                # Launch the experiments
                i_list = range(nb_repeat_exp)
                from joblib import parallel_backend
                with parallel_backend('threading',
                    n_jobs=nb_cores_used):
                Parallel(nb_cores_used)(delayed(main_function)
                    (i) for i in i_list)
```