A practical tutorial on Variational Bayes

Minh-Ngoc Tran, Trong-Nghia Nguyen, Viet-Hung Dao *

Abstract

This tutorial gives a quick introduction to Variational Bayes (VB), also called Variational Inference or Variational Approximation, from a practical point of view. The paper covers a range of commonly used VB methods and an attempt is made to keep the materials accessible to the wide community of data analysis practitioners. The aim is that the reader can quickly derive and implement their first VB algorithm for Bayesian inference with their data analysis problem. An end-user software package in Matlab together with the documentation can be found at `https://vbayeslab.github.io/VBLabDocs/`

Key words: Bayesian inference; Variational Inference; Neural Network; Bayesian Deep Learning.

1 Introduction

Bayesian inference has been long called for Bayesian computation techniques that are scalable to large data sets and applicable in big and complex models with a huge number of unknown parameters to infer. Sampling methods, such as Markov Chain Monte Carlo (MCMC) and Sequential Monte Carlo (SMC), in their current development do not meet this need. Sampling methods have not been successfully used in some modern areas such as deep neural networks. Even in more traditional areas such as graphical modelling and mixture modelling, it is very challenging to use MCMC and SMC. Variational Bayes (VB) is an optimization-based technique for approximate Bayesian inference, and provides a computationally efficient alternative to sampling methods. VB belongs to the bigger class of Variational Inference methods, which can also be used in the frequentist context for maximum likelihood estimation when there are missing data. The names Variational Bayes and Variational Inference are often used exchangeably in the literature, however, we prefer the former in this tutorial as we are solely interested in approximating the posterior distributions for Bayesian inference.

This tutorial provides a quick introduction to VB. There are many excellent tutorials and review papers on VB, however, most of them are either too abstract or tangential to the statistics readership, and do not offer much hands-on experience. This tutorial focuses on the practical aspect of VB, and is written to help the reader, who might even have a little background in computational statistics, be able to quickly learn about VB and implement the method to fit their model.

*Tran and Nguyen are with the University of Sydney Business School. Dao is with the UNSW Business School. Correspondence to minh-ngoc.tran@sydney.edu.au. The authors would like to thank David Nott and Emtiyaz Khan for useful comments and suggestions. We also thank Robert Salomone for pointing out many errors in an early version. Any errors left are our own.

Let y denote the data and $p(y|\theta)$ the likelihood function based on a postulated model, with $\theta \in \Theta$ the vector of model parameters to be estimated. Let $p(\theta)$ be the prior. Bayesian inference encodes all the available information about the model parameter θ in its posterior distribution with density

$$p(\theta|y) = \frac{p(y,\theta)}{p(y)} = \frac{p(\theta)p(y|\theta)}{p(y)} \propto p(\theta)p(y|\theta),$$

where $p(y) = \int_\Theta p(\theta)p(y|\theta)d\theta$, called the *marginal likelihood* or *evidence*. Here, the notation '$\propto$' means proportional up to the normalizing constant that is independent of the parameter (θ). In most Bayesian derivations, such a constant can be safely ignored. Bayesian inference typically requires computing expectations with respect to the posterior distribution. For example, the posterior mean, which is often used for point estimation, is an expectation of θ with respect to the posterior distribution $p(\theta|y)$. However, it is often difficult to compute such expectations, partly because the density $p(\theta|y)$ itself is intractable as the normalizing constant $p(y)$ is often unknown. For many applications, Bayesian inference is performed using MCMC, which estimates expectations w.r.t. $p(\theta|y)$ by sampling from it. For other applications where θ is high dimensional or fast computation is of primary interest, VB is an attractive alternative to MCMC. VB approximates the posterior distribution by a probability distribution with density $q(\theta)$ belonging to some tractable family of distributions $\mathcal{Q}$ such as Gaussians. The best VB approximation $q^* \in \mathcal{Q}$ is found by minimizing the Kullback-Leibler (KL) divergence *from* $q(\theta)$ *to* $p(\theta|y)$

$$q^* = \arg\min_{q\in\mathcal{Q}} \left\{ \mathrm{KL}\big(q\|p(\cdot|y)\big) := \int q(\theta)\log\frac{q(\theta)}{p(\theta|y)}d\theta \right\}. \tag{1}$$

Then, Bayesian inference is performed with the intractable posterior $p(\theta|y)$ replaced by the tractable VB approximation $q^*(\theta)$. It is easy to see that

$$\mathrm{KL}(q\|p(\cdot|y)) = -\int q(\theta)\log\frac{p(\theta)p(y|\theta)}{q(\theta)}d\theta + \log p(y),$$

thus minimizing KL is equivalent to maximizing the lower bound on $\log p(y)$[1]

$$\mathrm{LB}(q) := \int q(\theta)\log\frac{p(\theta)p(y|\theta)}{q(\theta)}d\theta = \mathbb{E}_q\Big(\log\frac{p(\theta)p(y|\theta)}{q(\theta)}\Big). \tag{2}$$

Without any constraint on $\mathcal{Q}$, the solution to (1) is $q^*(\theta)=p(\theta|y)$; of course this solution is useless as it is itself intractable. Depending on the constraint imposed on the class $\mathcal{Q}$, VB algorithms can be categorized into two classes: Mean Field VB (MFVB) and Fixed Form VB (FFVB) which are presented in Section 2 and Section 3, respectively. These two sections can be read completely separately depending on the reader's interest.

For researchers who wish to reproduce the numerical results in this tutorials, the Matlab code together with the data used in the examples are available on our gibhub `https://github.com/VBayesLab/Tutorial-on-VB`. For general practitioners, we provide an end-user software package VBLab, also available on our gibhub site, that allows users to easily perform approximate Bayesian inference in a wide range of statistical models. Section 4 describes this user-friendly VBLab software package and its applications.

[1]In this tutorial, the notation $a := b$ means a is defined by b. For any random variable or random vector X and any function $g(X)$, we denote by $\mathbb{E}_f\big(g(X)\big)$ (or $\mathbb{E}_{X\sim f}\big(g(X)\big)$, or simply $\mathbb{E}_X\big(g(X)\big)$) the expectation of $g(X)$ where X follows a probability distribution with density function $f(x)$.

2 Mean Field Variational Bayes

Let's write θ as $\theta=(\theta_1^\top,\theta_2^\top)^\top$. Here $a^\top$ denotes the transpose of vector a; and all vectors in this tutorial are column vectors. MFVB assumes the following factorization form for q

$$q(\theta)=q_1(\theta_1)q_2(\theta_2),$$

i.e., we ignore the posterior dependence between θ_1, θ_2 and attempt to approximate $p(\theta_1,\theta_2|y)$ by $q(\theta)=q_1(\theta_1)q_2(\theta_2)$. This is the only assumption/restriction we put on the class $\mathcal{Q}$. The lower bound in (2) is

$$\begin{aligned}
\mathrm{LB}(q_1,q_2) &= \int q_1(\theta_1)q_2(\theta_2)\log\frac{p(\theta,y)}{q_1(\theta_1)q_2(\theta_2)}d\theta_1 d\theta_2 \\
&= \int q_1(\theta_1)q_2(\theta_2)\log p(\theta,y)d\theta_1 d\theta_2 \\
&\quad -\int q_1(\theta_1)\log q_1(\theta_1)d\theta_1 - \int q_2(\theta_2)\log q_2(\theta_2)d\theta_2 \\
&= \int q_1(\theta_1)\mathbb{E}_{-\theta_1}[\log p(y,\theta)]d\theta_1 - \int q_1(\theta_1)\log q_1(\theta_1)d\theta_1 + C(q_2)
\end{aligned}$$

where $\mathbb{E}_{-\theta_1}[\log p(y,\theta)]:=\mathbb{E}_{q_2(\theta_2)}[\log p(y,\theta)]=\int q_2(\theta_2)\log p(y,\theta)d\theta_2$ and $C(q_2)$ is the term independent of q_1. The funny-looking notation $\mathbb{E}_{-\theta_1}(\cdot)$, meaning we take the expectation with respect to everything except θ_1, turns out to be very convenient when we deal with the general MFVB procedure later. Hence,

$$\begin{aligned}
\mathrm{LB}(q_1,q_2) &= \int q_1(\theta_1)\log\frac{\exp\left(\mathbb{E}_{-\theta_1}[\log p(y,\theta)]\right)}{q_1(\theta_1)}d\theta_1 + C(q_2) \\
&= \int q_1(\theta_1)\log\frac{\widetilde{q}_1(\theta_1)}{q_1(\theta_1)}d\theta_1 + C(q_2) + \log\widetilde{C}(q_2) \\
&= -\mathrm{KL}(q_1\|\widetilde{q}_1) + C(q_2) + \log\widetilde{C}(q_2), \qquad (3)
\end{aligned}$$

where $\widetilde{q}_1(\theta_1)$ is the probability density function determined by

$$\widetilde{q}_1(\theta_1):=\frac{\exp(\mathbb{E}_{-\theta_1}[\log p(y,\theta)])}{\widetilde{C}(q_2)}\propto\exp(\mathbb{E}_{-\theta_1}[\log p(y,\theta)]),$$

with $\widetilde{C}(q_2):=\int\exp(\mathbb{E}_{-\theta_1}[\log p(y,\theta)])d\theta_1$ also independent of q_1. We therefore have that

$$\mathrm{LB}(q_1,q_2) = -\mathrm{KL}(q_1\|\widetilde{q}_1) + \text{ constant independent of } q_1. \qquad (4)$$

Similarly,

$$\mathrm{LB}(q_1,q_2) = -\mathrm{KL}(q_2\|\widetilde{q}_2) + \text{ constant independent of } q_2, \qquad (5)$$

where $\widetilde{q}_2(\theta_2)\propto\exp(\mathbb{E}_{-\theta_2}[\log p(y,\theta)])$ with $\mathbb{E}_{-\theta_2}[\log p(y,\theta)]:=\int q_1(\theta_1)\log p(y,\theta)d\theta_1$. The expressions in (4)-(5) suggest a coordinate ascent optimization procedure for maximizing the lower bound: given

q_2, we minimize $\mathrm{KL}(q_1\|\widetilde{q}_1)$ to find q_1, and given q_1 we minimize $\mathrm{KL}(q_2\|\widetilde{q}_2)$ to find q_2. The hope is that solving the optimization problems

$$\min_{q_1}\big\{\mathrm{KL}(q_1\|\widetilde{q}_1)\big\} \quad \text{and} \quad \min_{q_2}\big\{\mathrm{KL}(q_2\|\widetilde{q}_2)\big\} \tag{6}$$

is easier than minimizing the original KL divergence between $q(\theta_1,\theta_2)$ and $p(\theta_1,\theta_2|y)$. If $\widetilde{q}_1$ and $\widetilde{q}_2$ are tractable and standard distributions[2], then of course the solution to (6) is $q_1=\widetilde{q}_1$ and $q_2=\widetilde{q}_2$. The most useful scenario is the case of *conjugate prior*: the prior $p(\theta_1)$ belongs to a parametric density family $\mathcal{F}_1$, then $\widetilde{q}_1(\theta_1)$ also belongs to $\mathcal{F}_1$. Similarly, the prior $p(\theta_2)$ belongs to a parametric density family $\mathcal{F}_2$, then $\widetilde{q}_2(\theta_2)$ also belongs to $\mathcal{F}_2$. Then the solutions to (6) are

$$q_1(\theta_1)=\widetilde{q}_1(\theta_1)\in\mathcal{F}_1 \quad \text{and} \quad q_2(\theta_2)=\widetilde{q}_2(\theta_2)\in\mathcal{F}_2,$$

and in order to identify q_1 and q_2 it's only necessary to compute their parameters. Computing the parameter in q_1 requires q_2 and vice versa, which suggests the following coordinate ascent-type algorithm for maximizing the lower bound:

Algorithm 1 (Mean Field Variational Bayes). *1. Initialize the parameter of $q_1(\theta_1)$*

2. Given $q_1(\theta_1)$, update the parameter of $q_2(\theta_2)$ using

$$q_2(\theta_2)\propto\exp\big(\mathbb{E}_{-\theta_2}[\log p(y,\theta)]\big)=\exp\Big(\int q_1(\theta_1)\log p(y,\theta_1,\theta_2)d\theta_1\Big). \tag{7}$$

3. Given $q_2(\theta_2)$, update the parameter of $q_1(\theta_1)$ using

$$q_1(\theta_1)\propto\exp\big(\mathbb{E}_{-\theta_1}[\log p(y,\theta)]\big)=\exp\Big(\int q_2(\theta_2)\log p(y,\theta_1,\theta_2)d\theta_2\Big). \tag{8}$$

4. Repeat Steps 2 and 3 until the stopping condition is met.

A stopping rule is to terminate the update if the change in the parameters of the VB posterior $q(\theta)=q_1(\theta_1)q_2(\theta_2)$ between two consecutive iterations is less than some threshold ϵ. In the case the lower bound $\mathrm{LB}(q_1,q_2)$ can be computed, one can stop the algorithm if the increase (or the percentage of the increase) in the lower bound is less than some threshold. Note that $\mathrm{LB}(q)$ increases after each iteration.

Example 2.1. Let $y=(11;12;8;10;9;8;9;10;13;7)$ be observations from $\mathcal{N}(\mu,\sigma^2)$, the normal distribution with mean μ and variance σ^2. Suppose that we use the prior $\mathcal{N}(\mu_0,\sigma_0^2)$ for μ and Inverse-Gamma(α_0,β_0) for σ^2, with hyperparameters $\mu_0=0$, $\sigma_0=10$, $\alpha_0=1$ and $\beta_0=1$. Assume the VB factorization $q(\mu,\sigma^2)=q(\mu)q(\sigma^2)$. Let's derive the MFVB procedure for approximating the posterior $p(\mu,\sigma^2|y)\propto p(\mu)p(\sigma^2)p(y|\mu,\sigma^2)$. We can view μ and σ^2 respectively as θ_1 and θ_2 in Algorithm 1.

[2]By a standard distribution, or a recognizable distribution, we mean a probability distribution that is well-understood and widely used, such as Gaussian, Gamma, etc. Yes, this definition of standard distribution isn't standard!

From (7), the optimal VB posterior for σ^2 is

$$\begin{aligned} q(\sigma^2) &\propto \exp\Big(\mathbb{E}_{-\sigma^2}[\log p(y,\mu,\sigma^2)]\Big) = \exp\Big(\mathbb{E}_{q(\mu)}[\log p(y,\mu,\sigma^2)]\Big) \\ &\propto \exp\Big(\mathbb{E}_{q(\mu)}[\log p(\sigma^2)+\log p(y|\mu,\sigma^2)]\Big) \\ &\propto \exp\Big(-(\alpha_0+\frac{n}{2}+1)\log\sigma^2-\big(\beta_0+\frac{1}{2}\mathbb{E}_{q(\mu)}[\sum(y_i-\mu)^2]\big)/\sigma^2\Big). \end{aligned}$$

In the above derivation, we have ignored all the constants independent of σ^2 as they are unnecessary for identifying the distribution $q(\sigma^2)$. It follows that $q(\sigma^2)$ is inverse-Gamma with parameters

$$\alpha_q = \alpha_0 + \frac{n}{2},\ \beta_q = \beta_0 + \frac{1}{2}\mathbb{E}_{q(\mu)}\Big[\sum(y_i-\mu)^2\Big].$$

Computation of the expecation $\mathbb{E}_{q(\mu)}(\cdot)$ becomes clear shortly after $q(\mu)$ is identified. From (8), the optimal VB posterior for μ is

$$\begin{aligned} q(\mu) &\propto \exp\Big(\mathbb{E}_{q(\sigma^2)}[\log p(y,\mu,\sigma^2)]\Big) \\ &\propto \exp\Big(\mathbb{E}_{q(\sigma^2)}[\log p(\mu)+\log p(y|\mu,\sigma^2)]\Big) \\ &\propto \exp\Big(-\frac{1}{2\sigma_0^2}(\mu^2-2\mu_0\mu)-\frac{n}{2}\mathbb{E}_{q(\sigma^2)}[\frac{1}{\sigma^2}](-2\bar{y}\mu+\mu^2)\Big) \\ &\propto \exp\Big(-\frac{1}{2}\underbrace{\big(\frac{1}{\sigma_0^2}+n\mathbb{E}_{q(\sigma^2)}[\frac{1}{\sigma^2}]\big)}_{A}\mu^2+\mu\underbrace{\big(\frac{\mu_0}{\sigma_0^2}+n\bar{y}\mathbb{E}_{q(\sigma^2)}[\frac{1}{\sigma^2}]\big)}_{B}\Big) \\ &= \exp\Big(-\frac{1}{2}A\mu^2+B\mu\Big) \\ &\propto \exp\Big(-\frac{1}{2}\frac{(\mu-B/A)^2}{1/A}\Big). \end{aligned}$$

It follows that $q(\mu)$ is Gaussian with mean μ_q and variance σ_q^2

$$\mu_q = \frac{\frac{\mu_0}{\sigma_0^2}+n\bar{y}\mathbb{E}_{q(\sigma^2)}[\frac{1}{\sigma^2}]}{\frac{1}{\sigma_0^2}+n\mathbb{E}_{q(\sigma^2)}[\frac{1}{\sigma^2}]},\ \ \sigma_q^2 = \Big(\frac{1}{\sigma_0^2}+n\mathbb{E}_{q(\sigma^2)}[\frac{1}{\sigma^2}]\Big)^{-1}.$$

With the distributions $q(\mu)$ and $q(\sigma^2)$ having identified, we are now able to compute the expectations w.r.t. $q(\mu)$ and $q(\sigma^2)$ in the above:

$$\begin{aligned} \beta_q &= \beta_0+\frac{1}{2}\mathbb{E}_{q(\mu)}\big[\sum(y_i-\mu)^2\big] \\ &= \beta_0+\frac{1}{2}\Big(\sum y_i^2-2n\bar{y}\mathbb{E}_{q(\mu)}[\mu]+n\mathbb{E}_{q(\mu)}[\mu^2]\Big) \\ &= \beta_0+\frac{1}{2}\sum y_i^2-n\bar{y}\mu_q+\frac{n}{2}(\mu_q^2+\sigma_q^2). \end{aligned}$$

As $q(\sigma^2)\sim$Inverse-Gamma(α_q,β_q), $\mathbb{E}(1/\sigma^2)=\alpha_q/\beta_q$. Hence,

$$\mu_q = \Big(\frac{\mu_0}{\sigma_0^2}+n\bar{y}\frac{\alpha_q}{\beta_q}\Big)/\Big(\frac{1}{\sigma_0^2}+n\frac{\alpha_q}{\beta_q}\Big),\ \text{ and }\ \sigma_q^2 = \Big(\frac{1}{\sigma_0^2}+n\frac{\alpha_q}{\beta_q}\Big)^{-1}.$$

Note that we did not make any assumption on the parametric form of optimal variational distributions $q(\mu)$ and $q(\sigma^2)$, it is the model (the prior and the likelihood) that determines their form. We arrive at the following updating procedure:

- Initialize μ_q,σ_q^2
- Update the following recursively

$$\begin{aligned}
\alpha_q &\leftarrow \alpha_0 + \frac{n}{2}, \\
\beta_q &\leftarrow \beta_0 + \frac{1}{2}\sum y_i^2 - n\bar{y}\mu_q + \frac{n}{2}(\mu_q^2 + \sigma_q^2), \\
\mu_q &\leftarrow \Big(\frac{\mu_0}{\sigma_0^2} + n\bar{y}\frac{\alpha_q}{\beta_q}\Big)/\Big(\frac{1}{\sigma_0^2} + n\frac{\alpha_q}{\beta_q}\Big), \\
\sigma_q^2 &\leftarrow \Big(\frac{1}{\sigma_0^2} + n\frac{\alpha_q}{\beta_q}\Big)^{-1},
\end{aligned}$$

  until convergence.

We can stop the iterative scheme when the change of the ℓ_2-norm of the vector $\lambda = (\alpha_q,\beta_q,\mu_q,\sigma_q^2)^\top$ is smaller than some ϵ, $\epsilon = 10^{-5}$ for example. We can also initialize α_q,β_q and then update the variational parameters recursively in the order of μ_q, σ_q^2, α_q and β_q. However, it's often a better idea to initialize μ_q,σ_q^2 as it is easier to guess the values related to location parameters than the scale parameters. Figure 1 plots the posterior densities estimated by the MFVB algorithm derived above, and by Gibbs sampling.

△

It is straightforward to extend the MFVB procedure in Algorithm 1 to the general case where θ is divided into k blocks $\theta = (\theta_1^\top,\theta_2^\top,...,\theta_k^\top)^\top$, and where we want to approximate the posterior $p(\theta_1,\theta_2,...,\theta_k|y)$ by $q(\theta) = q_1(\theta_1)q_2(\theta_2)...q_k(\theta_k)$. The optimal $q_j(\theta_j)$ that maximizes LB(q), when $q_1,...,q_{j-1},q_{j+1},...,q_k$ are fixed, is

$$q_j(\theta_j) \propto \exp\big(\mathbb{E}_{-\theta_j}[\log p(y,\theta)]\big), \quad j = 1,...,k. \tag{9}$$

Here $\mathbb{E}_{-\theta_j}(\cdot)$ denotes the expectation w.r.t. $q_1,...,\ q_{j-1},\ q_{j+1},...,\ q_k$, i.e.,

$$\mathbb{E}_{-\theta_j}\big[\log p(y,\theta)\big] := \int q_1(\theta_1)...q_{j-1}(\theta_{j-1})q_{j+1}(\theta_{j+1})...q_k(\theta_k)\log p(y,\theta)d\theta_1....d\theta_{j-1}d\theta_{j+1}...d\theta_k.$$

A similar procedure to Algorithm 1 can be developed, in which we first initialize the parameters in the $k-1$ factors $q_1,...,q_{k-1}$, then update q_k and the other factors recursively.

2.1 MFVB for elaborate models

One of the difficulties in using MFVB is that the optimal variational distributions in (9) sometimes do not admit a standard form. In Example 2.1, for example, if the data y_i does not follow a normal distribution but a Student's t distribution $t_\nu(\mu,\sigma^2)$, then it can be seen that the optimal variational distribution $q(\mu)$ does not have the form of a Gaussian distribution or any standard

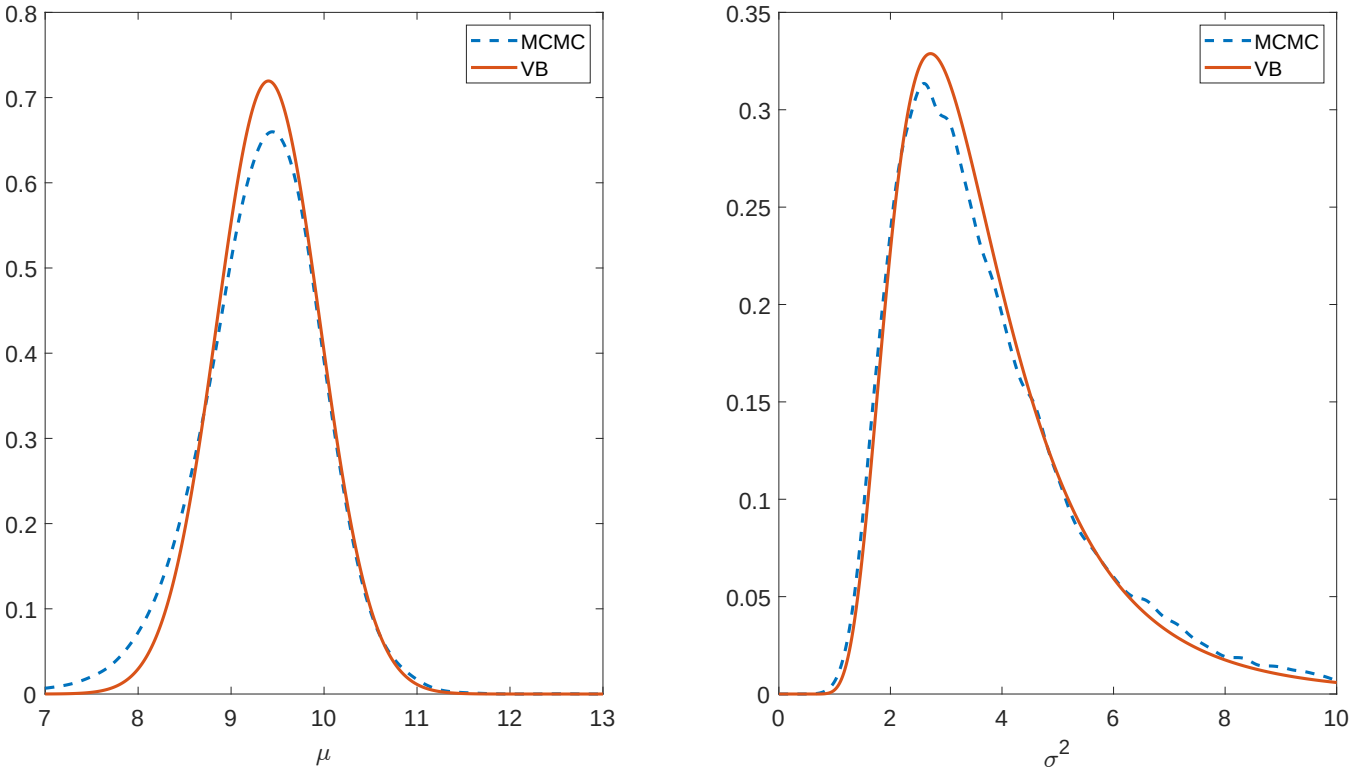

Figure 1: Example 2.1: Posterior density for μ and σ^2 estimated by MFVB and Gibbs sampling. The CPU time taken by VB was 0.006 seconds, by the Gibbs sampling scheme was 1.81 seconds. VB was about 300 times faster.

probability distribution. In some situations, however, by introducing auxiliary variables, we can equivalently represent the model by augmenting the parameter space such that MFVB is applicable. The use of auxiliary variables to facilitate statistical computations is widely used in many areas of statistics. We follow Wand et al. (2011) and use the term *elaborate model* to refer to a statistical model in which its prior or its data density can be augmented using auxiliary variables such that the optimal variational distributions in (9) admit a standard form. Introducing auxiliary variables makes MFVB tractable, but this might come at the price of reducing the variational approximation accuracy; however, we won't discuss this issue in any detail in this tutorial.

More precisely, consider the standard Bayesian model

$$y|\theta \sim p(y|\theta), \qquad \theta \sim p(\theta). \tag{10}$$

Suppose that there exists an auxiliary variable η such that

$$p(y|\theta) = \int p(y|\theta,\eta)p(\eta|\theta)\mathrm{d}\eta, \tag{11}$$

then model (10) can be equivalently represented as

$$y|\theta,\eta \sim p(y|\theta,\eta), \qquad \eta|\theta \sim p(\eta|\theta), \qquad \theta \sim p(\theta). \tag{12}$$

The model (10) is said to be elaborate if it can be presented as the hierarchical model (12) and, under the variational factorization $q(\theta,\eta)=q(\theta)q(\eta)$, the optimal variational distributions $q(\theta)$ and

$q(\eta)$ in (9) admit a standard form. The idea of elaborate models applies to the prior too, in which one can represent the prior $p(\theta)$ in a hierarchical form using auxiliary variables.

We now demonstrate this idea in the Bayesian Lasso model. Consider the linear regression problem

$$y = \mu 1_n + X\beta + \epsilon,$$

where y is the vector of responses, X is the $n \times p$ matrix of covariates, 1_n is the $n \times 1$ vector of 1s, and ϵ is the vector of i.i.d. normal errors $\mathcal{N}(0,\sigma^2)$. Without loss of generality, we assume that y and X have been centered so that μ is zero and omitted from the model. Regression analysis is often concerned with estimating $\beta = (\beta_1,...,\beta_p)^\top$ and simultaneously identifying the important covariates. The least absolute shrinkage and selection operator (Lasso) method solves this problem by minimizing the sum of squared errors and a regularization term

$$\min_\beta \Big\{(y - X\beta)'(y - X\beta) + \widetilde{\lambda}\sum_{j=1}^{p} |\beta_j|\Big\}, \tag{13}$$

where $\widetilde{\lambda}>0$ is the tuning parameter controlling the amount of regularization. The Lasso estimator, i.e. the solution of (13), can be interpreted as the posterior mode in a Bayesian context where a conditional Laplace prior is used for β

$$p(\beta|\sigma^2) = \prod_{j=1}^{p} \frac{\lambda}{2\sqrt{\sigma^2}} e^{-\lambda|\beta_j|/\sqrt{\sigma^2}}, \tag{14}$$

for some shrinkage parameter[3] λ. The posterior mode of β is the Lasso estimator in (13) with $\widetilde{\lambda}=2\sqrt{\sigma^2}\lambda$.

It is difficult to use MFVB for approximating the posterior $p(\beta,\sigma^2|X,y)$ in this case, as the optimal conditional variational distribution of β does not admit a standard form. However, it turns out that we can use auxiliary variables to make this Bayesian model elaborate and overcome the aforementioned difficulty.

It is well-known that a Laplace distribution can be represented as a mixture of normal and exponential distributions as follows

$$\frac{\lambda}{2} e^{-\lambda|z|} = \int_0^\infty \frac{1}{\sqrt{2\pi s}} e^{-z^2/(2s)} \frac{\lambda^2}{2} e^{-\lambda^2 s/2} \mathrm{ds}.$$

Using this representation, after some algebra, we have that

$$\frac{\lambda}{2\sqrt{\sigma^2}} e^{-\lambda|\beta_j|/\sqrt{\sigma^2}} = \int_0^\infty \frac{1}{\sqrt{2\pi\sigma^2\tau}} e^{-\beta_j^2/(2\sigma^2\tau)} \frac{\lambda^2}{2} e^{-\lambda^2\tau/2} \mathrm{d}\tau.$$

This motivates the following hierarchical representation of the Bayesian Lasso model

$$\begin{aligned} y|X,\beta,\sigma^2 &\sim \mathcal{N}(X\beta, \sigma^2 I_n), \\ \beta_j|\sigma^2,\tau_j &\sim \mathcal{N}(0, \sigma^2\tau_j), \\ \tau_j &\sim \mathrm{Exp}\big(\frac{\lambda^2}{2}\big) = \frac{\lambda^2}{2} e^{-\lambda^2\tau_j/2}, \quad j = 1,...,p. \end{aligned}$$

[3]This parameter shouldn't be confused with the variational parameter λ in Section 3.

The conjugate prior for σ^2 is inverse Gamma and we use the improper prior $p(\sigma^2) \propto 1/\sigma^2$ in this example. The shrinkage parameter λ can be selected in some way, here we use a full Bayesian treatement and put a Gamma prior on λ^2

$$p(\lambda^2) = \frac{\delta^r}{\Gamma(r)}(\lambda^2)^{r-1}e^{-\delta\lambda^2},$$

with r and δ hyperparameters and pre-specified. Note that we use a prior for λ^2, not λ, as this leads to a tractable form for the optimal conditional variational distribution for λ^2.

The model parameters include β, $\tau=(\tau_1,...,\tau_p)^\top$, σ^2 and λ^2. Let us use the following mean field variational distribution

$$q(\beta, \tau, \sigma^2, \lambda^2) = q(\beta)q(\tau)q(\sigma^2)q(\lambda^2).$$

With this factorization, all the optimal conditional variational distributions admit a standard form. The optimal variational distribution for β is $\mathcal{N}(\mu_\beta,\Sigma_\beta)$ with

$$\mu_\beta = \big(X^\top X + D_\tau\big)^{-1}X^\top y, \quad \Sigma_\beta = \big(X^\top X + D_\tau\big)^{-1}/\mathbb{E}_q(\frac{1}{\sigma^2}),$$

where $D_\tau := \mathrm{diag}\big(\mathbb{E}_q(1/\tau_1),\cdots,\mathbb{E}_q(1/\tau_p)\big)$. Here $\mathbb{E}_q(\cdot)$ denotes expectation with respect to the variational distribution q. The optimal variational distributions for τ_j are independent of each other, where $\widetilde{\tau}_j := 1/\tau_j$ follows an inverse-Gaussian with location and scale parameters

$$\mu_{\widetilde{\tau}_j} = \Big(\frac{\mathbb{E}_q(\lambda^2)}{\mathbb{E}_q\big(\beta_j^2/\sigma^2\big)}\Big)^{1/2}, \quad \lambda_{\widetilde{\tau}_j} = \mathbb{E}_q(\lambda^2).$$

The optimal distribution for σ^2 is inverse Gamma with the parameters

$$\alpha_{\sigma^2} = \frac{1}{2}(n+p), \quad \beta_{\sigma^2} = \frac{1}{2}\mathbb{E}_q\|y - X\beta\|^2 + \frac{1}{2}\sum_{j=1}^{p}\mathbb{E}_q\big(\frac{\beta_j^2}{\tau_j}\big).$$

Finally, the optimal variatinoal distribution for λ^2 is Gamma with

$$\alpha_{\lambda^2} = r + 1, \quad \beta_{\lambda^2} = \delta + \frac{1}{2}\sum_j \mathbb{E}_q(\tau_j).$$

Using the results regarding to the moments of these standard distributions, we have

$$\mathbb{E}_q\big(\frac{1}{\tau_j}\big) = \mathbb{E}_q(\widetilde{\tau}_j) = \mu_{\widetilde{\tau}_j}, \qquad \mathbb{E}_q(\tau_j) = \mathbb{E}_q\big(\frac{1}{\widetilde{\tau}_j}\big) = \frac{1}{\mu_{\widetilde{\tau}_j}} + \frac{1}{\lambda_{\widetilde{\tau}_j}},$$
$$\mathbb{E}_q\big(\frac{1}{\sigma^2}\big) = \frac{\alpha_{\sigma^2}}{\beta_{\sigma^2}}, \qquad \mathbb{E}_q\big(\lambda^2\big) = \frac{\alpha_{\lambda^2}}{\beta_{\lambda^2}},$$
$$\mathbb{E}_q(\beta_j^2) = \mu_{\beta,j}^2 + \Sigma_{\beta,jj},$$

where $\mu_{\beta,j}$ is the jth element of vector μ_β and $\Sigma_{\beta,jj}$ is the (j,j) element of matrix Σ_β. We arrive at the MFVB procedure for Bayesian inference in the Bayesian Lasso model.

Algorithm 2 (MFVB for Bayesian Lasso)**.** *Initialize α_{σ^2}, β_{σ^2}, $\mu_{\widetilde{\tau}_j}$ and $\lambda_{\widetilde{\tau}_j}$, $j=1,...,p$, then update the following until convergence:*

True β	μ_β
3	3.0029 (0.0042)
1.5	1.4946 (0.0041)
0	0.0044 (0.0040)
0	0.0074 (0.0043)
2	2.0064 (0.0041)
0	−0.0088 (0.0044)
0	−0.0007 (0.0041)
0	0.0008 (0.0039)

Table 1: Example 2.2: The performance of MFVB for the Bayesian Lasso model. The first column lists the true β and the second column lists the point estimate μ_β (at convergence) of the posterior mean of β, with the estimates of the posterior standard deviations in brackets.

- *Update* μ_β *and* Σ_β

$$\mu_\beta = \big(X^\top X + D_\tau\big)^{-1} X^\top y, \quad \Sigma_\beta = \frac{\beta_{\sigma^2}}{\alpha_{\sigma^2}} \big(X^\top X + D_\tau\big)^{-1},$$

  where $D_\tau := \mathrm{diag}\big(\mu_{\tilde{\tau}_1}, \cdots, \mu_{\tilde{\tau}_p}\big)$.

- *Update* α_{λ^2} *and* β_{λ^2}

$$\alpha_{\lambda^2} = r + 1, \quad \beta_{\lambda^2} = \delta + \frac{1}{2} \sum_j \Big(\frac{1}{\mu_{\tilde{\tau}_j}} + \frac{1}{\lambda_{\tilde{\tau}_j}} \Big).$$

- *Update* $\mu_{\tilde{\tau}_j}$ *and* $\lambda_{\tilde{\tau}_j}$, $j = 1, \ldots, p$

$$\mu_{\tilde{\tau}_j} = \Big(\frac{\alpha_{\lambda^2}/\beta_{\lambda^2}}{\big(\alpha_{\sigma^2}/\beta_{\sigma^2}\big)\big(\mu^2_{\beta,j} + \Sigma_{\beta,jj}\big)} \Big)^{1/2}, \quad \lambda_{\tilde{\tau}_j} = \frac{\alpha_{\lambda^2}}{\beta_{\lambda^2}}.$$

- *Update* α_{σ^2} *and* β_{σ^2}

$$\alpha_{\sigma^2} = \frac{1}{2}(n+p), \quad \beta_{\sigma^2} = \frac{1}{2}\|y - X\mu_\beta\|^2 + \frac{1}{2}\mathrm{tr}(X\Sigma_\beta X^\top) + \frac{1}{2}\sum_{j=1}^{p} (\mu^2_{\beta,j} + \Sigma_{\beta,jj})\mu_{\tilde{\tau}_j}.$$

Example 2.2 (Bayesian Lasso)**.** A data set of size $n = 500$ is generated from the model

$$y = x^\top \beta + \sigma\epsilon,$$

where $\beta = (3,\ 1.5,\ 0,\ 0,\ 2,\ 0,\ 0,\ 0)^\top$, $x_j \sim \mathcal{N}(0,1)$, $\epsilon \overset{iid}{\sim} \mathcal{N}(0,1)$ and $\sigma = 0.1$.

The MFVB algorithm 2 stops after 22 iterations when the l_2 difference between two consecutive updates of μ_β is less than $1e{-}10$. The hyperparameters r and δ are set to 0. Table 1 summarizes the result, Figure 2 plots the updates of μ_β over iterations.

△

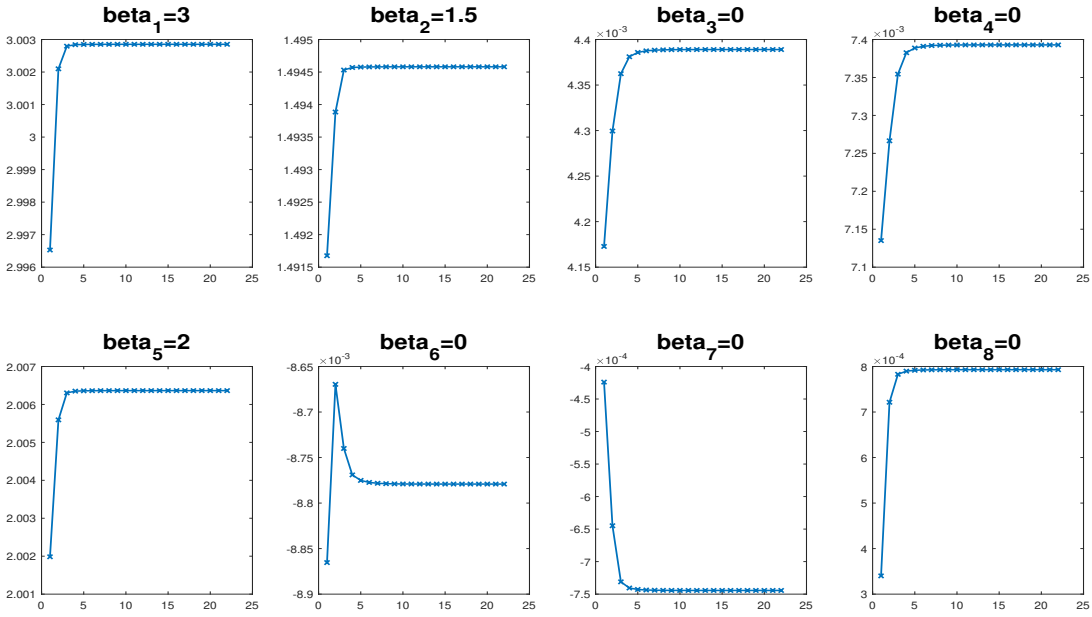

Figure 2: Example 2.2: The updates of μ_β over iterations.

2.2 Some remarks about MFVB

Early work on MFVB in machine learning and statistics can be found in Waterhouse et al. (1996); Jordan et al. (1999) and Titterington (2004), and tutorial-style introductions to MFVB can be found in Bishop (2006) and Ormerod and Wand (2010). MFVB has been successfully used in some statistical areas such as mixture modelling and graphical modelling. In mixture modelling, for example, MFVB does not only offer a fast Bayesian estimation method, but is also able to deal with the challenging model selection problem in a convenient way. See, e.g., Ghahramani and Hinton (2000); Corduneanu and Bishop (2001); McGrory and Titterington (2007); Giordani et al. (2013) and Tran et al. (2014). The approximation accuracy and large-scale properties of MFVB have been extensively studied recently, its cover requires a book-length discussion and is omitted in this tutorial.

3 Fixed Form Variational Bayes

FFVB assumes a fixed parametric form for the VB approximation density q, i.e. $q = q_\lambda$ belongs to some class of distributions $\mathcal{Q}$ indexed by a vector λ called the *variational parameter*. For example, q_λ is a Gaussian distribution with mean μ and covariance matrix Σ. FFVB finds the best q_λ in the class $\mathcal{Q}$ by optimizing the lower bound

$$\mathrm{LB}(\lambda) := \mathrm{LB}(q_\lambda) = \mathbb{E}_{q_\lambda}\left[\log \frac{p(\theta)p(y|\theta)}{q_\lambda(\theta)}\right] = \mathbb{E}_{q_\lambda}\big[h_\lambda(\theta)\big], \tag{15}$$

with

$$h_\lambda(\theta) := \log\big(\frac{p(\theta)p(y|\theta)}{q_\lambda(\theta)}\big).$$

Later we also use $h(\theta)$, without the subscript, to denote the model-specific function $\log\big(p(\theta)p(y|\theta)\big)$. Except for a few trivial cases where the LB can be computed analytically and optimized using classical optimization routines, stochastic optimization is often used to optimize $\mathrm{LB}(\lambda)$. The gradient vector of LB is

$$\begin{aligned}
\nabla_\lambda \mathrm{LB}(\lambda) &= \int_\Theta \nabla_\lambda q_\lambda(\theta) \log\frac{p(\theta)p(y|\theta)}{q_\lambda(\theta)} d\theta - \int_\Theta q_\lambda(\theta)\nabla_\lambda \log q_\lambda(\theta) d\theta \\
&= \int_\Theta q_\lambda(\theta)\nabla_\lambda \log q_\lambda(\theta) \log\frac{p(\theta)p(y|\theta)}{q_\lambda(\theta)} d\theta - \int_\Theta \nabla_\lambda q_\lambda(\theta) d\theta \\
&= \int_\Theta q_\lambda(\theta)\nabla_\lambda \log q_\lambda(\theta) \log\frac{p(\theta)p(y|\theta)}{q_\lambda(\theta)} d\theta - \nabla_\lambda \int_\Theta q_\lambda(\theta) d\theta \\
&= \mathbb{E}_{q_\lambda}\left[\nabla_\lambda \log q_\lambda(\theta) \times \log\frac{p(\theta)p(y|\theta)}{q_\lambda(\theta)}\right] \\
&= \mathbb{E}_{q_\lambda}\left[\nabla_\lambda \log q_\lambda(\theta) \times h_\lambda(\theta)\right]. \qquad (16)
\end{aligned}$$

The gradient in this form is often referred to as *score-function gradient*, another way known as *reparameterization gradient* to compute the gradient of the lower bound is discussed later in (24). It follows from (16) that, by generating[4] $\theta \sim q_\lambda(\theta)$, it is straightforward to obtain an unbiased estimator $\widehat{\nabla_\lambda \mathrm{LB}}(\lambda)$ of the gradient $\nabla_\lambda \mathrm{LB}(\lambda)$, i.e., $\mathbb{E}\big[\widehat{\nabla_\lambda \mathrm{LB}}(\lambda)\big] = \nabla_\lambda \mathrm{LB}(\lambda)$. Therefore, we can use stochastic optimization[5] to optimize $\mathrm{LB}(\lambda)$. The basic algorithm is as follows:

Algorithm 3 (Basic FFVB algorithm)**.**

- *Initialize $\lambda^{(0)}$ and stop the following iteration if the stopping criterion is met.*
- *For $t = 0,1,...$*
  - *Generate $\theta_s \sim q_{\lambda^{(t)}}(\theta)$, $s = 1,...,S$*
  - *Compute the unbiased estimate of the LB gradient*

$$\widehat{\nabla_\lambda LB}(\lambda^{(t)}) := \frac{1}{S}\sum_{s=1}^{S} \nabla_\lambda \log q_\lambda(\theta_s) \times h_\lambda(\theta_s)|_{\lambda=\lambda^{(t)}}.$$

  - *Update*

$$\lambda^{(t+1)} = \lambda^{(t)} + a_t \widehat{\nabla_\lambda LB}(\lambda^{(t)}). \qquad (17)$$

The algorithmic parameter S is referred to as the number of Monte Carlo samples (used to estimate the gradient of the lower bound). The sequence of learning rates $\{a_t\}$ should satisfy the theoretical requirements $a_t > 0$, $\sum_t a_t = \infty$ and $\sum_t a_t^2 < \infty$. However, this basic VB algorithm hardly works in practice and requires some refinements to make it work. Much of the rest of this section focuses on presenting and explaining those refinements.

[4] In Monte Carlo simulation, by $\theta \sim q_\lambda(\theta)$ we mean that we draw a random variable or random vector θ from the probability distribution with density $q_\lambda(\theta)$. That notation also means θ is a random variable/vector whose probability density function is $q_\lambda(\theta)$.

[5] Unbiased estimate of the gradient of the target function is theoretically required in stochastic optimization.

3.1 Stopping criterion

Let us first discuss on the stopping rule. An easy-to-implement stopping rule is to terminate the updating procedure if the change between $\lambda^{(t+1)}$ and $\lambda^{(t)}$, e.g. in terms of the Euclidean distance, is less than some threshold ϵ. However, it is difficult to select a meaningful ϵ as such a distance depends on the scales and the length of the vector λ. Denote by $\widehat{\mathrm{LB}}(\lambda)$ an estimate of $\mathrm{LB}(\lambda)$ by sampling from $q_\lambda(\theta)$, i.e.,

$$\widehat{\mathrm{LB}}(\lambda) = \frac{1}{S}\sum_{s=1}^{S} h_\lambda(\theta_s), \quad \theta_s \sim q_\lambda(\theta).$$

Although $\mathrm{LB}(\lambda)$ is expected to be non-decreasing over iterations, its sample estimate $\widehat{\mathrm{LB}}(\lambda)$ might not be. To account for this, we can use a moving average of the lower bounds over a window of t_W iterations, $\overline{\mathrm{LB}}(\lambda^{(t)}) = (1/t_W)\sum_{k=1}^{t_W}\widehat{\mathrm{LB}}(\lambda^{(t-k+1)})$. At convergence, the values $\mathrm{LB}(\lambda^{(t)})$ stay roughly the same, therefore $\overline{\mathrm{LB}}(\lambda^{(t)})$ will average out the noise in $\widehat{\mathrm{LB}}(\lambda^{(t)})$ and is stable. The stopping rule that is widely used in machine learning is to stop training if the moving averaged lower bound does not improve after P iterations; and P is sometimes fancily referred to as the *patience* parameter. Typical choice is $P=20$ or $P=50$, and $t_W=20$ or $t_W=50$. Note that, we must not use the last $\lambda^{(t)}$ as the final estimate of λ, but the one corresponding to the largest $\overline{\mathrm{LB}}(\lambda^{(t)})$.

3.2 Adaptive learning rate and natural gradient

Let's write the update in (17) as

$$\begin{cases} \lambda_1^{(t+1)} = \lambda_1^{(t)} + a_t \widehat{\nabla_{\lambda_1}\mathrm{LB}}(\lambda^{(t)}) \\ \dots \\ \lambda_{d_\lambda}^{(t+1)} = \lambda_{d_\lambda}^{(t)} + a_t \widehat{\nabla_{\lambda_{d_\lambda}}\mathrm{LB}}(\lambda^{(t)}), \end{cases}$$

with d_λ the size of vector λ, which shows that a common scalar learning rate a_t is used for all the coordinates of λ. Intuitively, each coordinate of vector $\lambda^{(t+1)}$ might need a different learning rate that can take into account the scale of that coordinate or the geometry of the space λ living in. It turns out that the basic Algorithm 3 rarely works in practice without a method for selecting the learning rate adaptively.

3.2.1 Adaptive learning rate

For a coordinate i with a large variance $\mathbb{V}(\widehat{\nabla_{\lambda_i}\mathrm{LB}}(\lambda^{(t)}))$, its learning rate $a_{t,i}$ should be small, otherwise the new update $\lambda_i^{(t+1)}$ jumps all over the place and destroys everything the process has learned so far. Denote $g_t := \widehat{\nabla_\lambda \mathrm{LB}}(\lambda^{(t)})$ be the gradient vector at step t, and $v_t := (g_t)^2$ (this is a coordinate-wise operator). The commonly used adaptive learning rate methods such as ADAM and AdaGrad work by scaling the coordinates of g_t by their corresponding variances. These variances are estimated by moving average. The algorithm below is a basic version of this class of adaptive learning methods:

1) Initialize $\lambda^{(0)}$, g_0 and v_0 and set $\bar{g}=g_0$, $\bar{v}=v_0$. Let $\beta_1,\beta_2\in(0,1)$ be adaptive learning weights.

2) For $t=0,1,...$, update

$$\begin{aligned} \bar{g} &= \beta_1 \bar{g} + (1-\beta_1) g_t \\ \bar{v} &= \beta_2 \bar{v} + (1-\beta_2) v_t \\ \lambda^{(t+1)} &= \lambda^{(t)} + \alpha_t \bar{g}/\sqrt{\bar{v}}, \end{aligned}$$

with α_t a scalar step size. Here $\bar{g}/\sqrt{\bar{v}}$ should be understood component wise.

Note that the LB gradients g_t have also been smoothened out using moving average. This helps to accelerate the convergence - a method known as the momentum method in the stochastic optimization literature. Typical choice of the scalar α_t is

$$\alpha_t = \min\left(\epsilon_0, \epsilon_0 \frac{\tau}{t}\right) = \begin{cases} \epsilon_0, & t \le \tau \\ \epsilon_0 \frac{\tau}{t}, & t > \tau \end{cases} \tag{18}$$

for some small *fixed learning rate* ϵ_0 (e.g. 0.1 or 0.01) and some threshold τ (e.g., 1000). In the first τ iterations, the training procedure explores the learning space with a fixed learning rate ϵ_0, then this exploration is settled down by reducing the step size after τ iterations.

3.2.2 Natural gradient

Natural gradient can be considered as an adaptive learning method that exploits the geometry of the λ space. The ordinary gradient $\nabla_\lambda \mathrm{LB}(\lambda)$ does not adequately capture the geometry of the approximating family $\mathcal{Q}$ of $q_\lambda(\theta)$. A small Euclidean distance between λ and λ' does not necessarily mean a small KL divergence between $q_\lambda(\theta)$ and $q_{\lambda'}(\theta)$. Statisticians and machine learning researchers have long realized the importance of information geometry on the manifold of a statistical model, and that the steepest direction for optimizing the objective function $\mathrm{LB}(\lambda)$ on the manifold formed by the family $q_\lambda(\theta)$ is directed by the so-called natural gradient which is defined by pre-multiplying the ordinary gradient with the inverse of the Fisher information matrix

$$\nabla_\lambda \mathrm{LB}(\lambda)^{\mathrm{nat}} := I_F^{-1}(\lambda) \nabla_\lambda \mathrm{LB}(\lambda),$$

with $I_F(\lambda) = \mathrm{cov}_{q_\lambda}(\nabla_\lambda \log q_\lambda(\theta))$ the Fisher information matrix about λ with respect to the distribution q_λ. Given an unbiased estimate $\widehat{\nabla_\lambda \mathrm{LB}}(\lambda)$, the unbiased estimate of the natural gradient is

$$\widehat{\nabla_\lambda \mathrm{LB}}(\lambda)^{\mathrm{nat}} = I_F^{-1}(\lambda) \widehat{\nabla_\lambda \mathrm{LB}}(\lambda). \tag{19}$$

The main difficulty in using the natural gradient is the computation of $I_F(\lambda)$, and the solution of the linear systems required to compute (19). The problem is more severe in high dimensional models because this matrix has a large size. An efficient method for computing $I_F(\lambda)^{-1}\widehat{\nabla_\lambda \mathrm{LB}}(\lambda)$ is using iterative conjugate gradient methods which solve the linear system $I_F(\lambda)x = \widehat{\nabla_\lambda \mathrm{LB}}(\lambda)$ for x using only matrix-vector products involving $I_F(\lambda)$. In some cases this matrix vector product can be done efficiently both in terms of computational time and memory requirements by exploiting the structure of the Fisher matrix $I_F(\lambda)$. See Section 3.5.2 for a special case where the natural gradient is computed efficiently in high dimensional problems.

As mentioned before, the gradient momentum method is often useful in stochastic optimization that helps accelerate and stabilize the optimization procedure. The momentum update rule with the natural gradient is

$$\begin{aligned}\overline{\nabla_\lambda \mathrm{LB}} &= \alpha_\mathrm{m}\overline{\nabla_\lambda \mathrm{LB}} + (1-\alpha_\mathrm{m})\widehat{\nabla_\lambda \mathrm{LB}}(\lambda^{(t)})^{\mathrm{nat}},\\ \lambda^{(t+1)} &= \lambda^{(t)} + \alpha_t \overline{\nabla_\lambda \mathrm{LB}},\end{aligned}$$

where $\alpha_\mathrm{m} \in [0,1]$ is the momentum weight; α_m around 0.6-0.9 is a typical choice. The use of the moving average gradient $\overline{\nabla_\lambda \mathrm{LB}}$ also helps remove some of the noise inherent in the estimated gradients of the lower bound. Note that the momentum method is already embedded in the moving-average-based adaptive learning rate methods in Section 3.2.1.

3.3 Control variate

As is typical of stochastic optimization algorithms, the performance of Algorithm 3 depends greatly on the variance of the noisy gradient. Variance reduction for the noisy gradient is a key ingredient in FFVB algorithms. This section describes a control variate technique for variance reduction, another technique known as *reparameterization trick* is presented in Section 3.4.

Let $\theta_s \sim q_\lambda(\theta)$, $s=1,...,S$, be S samples from the variational distribution $q_\lambda(\theta)$. A naive estimator of the ith element of the vector $\nabla_\lambda \mathrm{LB}(\lambda)$ is

$$\widehat{\nabla_{\lambda_i} \mathrm{LB}}(\lambda)^{\mathrm{naive}} = \frac{1}{S}\sum_{s=1}^{S} \nabla_{\lambda_i}[\log q_\lambda(\theta_s)] \times h_\lambda(\theta_s), \tag{20}$$

whose variance is often too large to be useful. For any number c_i, consider

$$\widehat{\nabla_{\lambda_i} \mathrm{LB}}(\lambda) = \frac{1}{S}\sum_{s=1}^{S} \nabla_{\lambda_i}[\log q_\lambda(\theta_s)](h_\lambda(\theta_s) - c_i), \tag{21}$$

which is still an unbiased estimator of $\nabla_{\lambda_i}\mathrm{LB}(\lambda)$ since $\mathbb{E}(\nabla_\lambda[\log q_\lambda(\theta)]) = 0$, whose variance can be greatly reduced by an appropriate choice of control variate c_i. The variance of $\widehat{\nabla_{\lambda_i} \mathrm{LB}}(\lambda)$ is

$$\frac{1}{S}\mathbb{V}\Big(\nabla_{\lambda_i}[\log q_\lambda(\theta)]h_\lambda(\theta)\Big) + \frac{c_i^2}{S}\mathbb{V}\Big(\nabla_{\lambda_i}[\log q_\lambda(\theta)]\Big) - \frac{2c_i}{S}\mathrm{cov}\Big(\nabla_{\lambda_i}[\log q_\lambda(\theta)]h_\lambda(\theta), \nabla_{\lambda_i}[\log q_\lambda(\theta)]\Big).$$

The optimal c_i that minimizes this variance is

$$c_i = \mathrm{cov}\Big(\nabla_{\lambda_i}[\log q_\lambda(\theta)]h_\lambda(\theta), \nabla_{\lambda_i}[\log q_\lambda(\theta)]\Big) \Big/ \mathbb{V}\Big(\nabla_{\lambda_i}[\log q_\lambda(\theta)]\Big). \tag{22}$$

Then $\mathbb{V}(\widehat{\nabla_{\lambda_i} \mathrm{LB}}(\lambda)) = \mathbb{V}(\widehat{\nabla_{\lambda_i} \mathrm{LB}}(\lambda)^{\mathrm{naive}})(1-\rho_i^2) \leq \mathbb{V}(\widehat{\nabla_{\lambda_i} \mathrm{LB}}(\lambda)^{\mathrm{naive}})$, where ρ_i is the correlation between $\nabla_{\lambda_i}[\log q_\lambda(\theta)]h_\lambda(\theta)$ and $\nabla_{\lambda_i}[\log q_\lambda(\theta)]$. Often, ρ_i^2 is very close to 1, which leads to a large variance reduction.

One can estimate the numbers c_i in (22) using samples $\theta_s \sim q_\lambda(\theta)$. In order to ensure the unbiasedness of the gradient estimator, the samples used to estimate c_i must be independent of the samples used to estimate the gradient. In practice, the c_i can be updated sequentially as follows.

At iteration t, we use the c_i computed in the previous iteration $t-1$, i.e. based on the samples from $q_{\lambda^{(t-1)}}(\theta)$, to estimate the gradient $\widehat{\nabla_\lambda \mathrm{LB}}(\lambda^{(t)})$, which is computed using new samples from $q_{\lambda^{(t)}}(\theta)$. We then update the c_i using this new set of samples. By doing so, the unbiasedness is guaranteed while no extra samples are needed in updating the control variates c_i.

Algorithm 4 provides a detailed pseudo-code implementation of the FFVB approach that uses the control variate for variance reduction and moving average adaptive learning, and Algorithm 5 implements the FFVB approach that uses the control variate and natural gradient.

Algorithm 4 (FFVB with control variates and adaptive learning)**. Input**: *Initial* $\lambda^{(0)}$*, adaptive learning weights* $\beta_1,\beta_2 \in (0,1)$*, fixed learning rate* ϵ_0*, threshold* τ*, rolling window size* t_W *and maximum patience* P*.* **Model-specific requirement**: *function* $h(\theta) := \log\big(p(\theta)p(y|\theta)\big)$*.*

- *Initialization*
  - *Generate* $\theta_s \sim q_{\lambda^{(0)}}(\theta)$*,* $s=1,...,S$*.*
  - *Compute the unbiased estimate of the LB gradient*

$$\widehat{\nabla_\lambda LB}(\lambda^{(0)}) := \frac{1}{S}\sum_{s=1}^{S} \nabla_\lambda \log q_\lambda(\theta_s) \times h_\lambda(\theta_s)|_{\lambda=\lambda^{(0)}}.$$

  - *Set* $g_0 := \widehat{\nabla_\lambda LB}(\lambda^{(0)})$*,* $v_0 := (g_0)^2$*,* $\bar{g} := g_0$*,* $\bar{v} := v_0$*.*
  - *Estimate the vector of control variates* c *as in* (22) *using the samples* $\{\theta_s, s=1,...,S\}$*.*
  - *Set* $t=0$*, patience*$=0$ *and* `stop=false`*.*
- *While* `stop=false`*:*
  - *Generate* $\theta_s \sim q_{\lambda^{(t)}}(\theta)$*,* $s=1,...,S$*.*
  - *Compute the unbiased estimate of the LB gradient*[6]

$$g_t := \widehat{\nabla_\lambda LB}(\lambda^{(t)}) = \frac{1}{S}\sum_{s=1}^{S} \nabla_\lambda \log q_\lambda(\theta_s) \circ \big(h_\lambda(\theta_s) - c\big)|_{\lambda=\lambda^{(t)}}.$$

  - *Estimate the new control variate vector* c *as in* (22) *using the samples* $\{\theta_s, s=1,...,S\}$*.*
  - *Compute* $v_t = (g_t)^2$ *and*

$$\bar{g} = \beta_1 \bar{g} + (1-\beta_1) g_t,\ \bar{v} = \beta_2 \bar{v} + (1-\beta_2) v_t.$$

  - *Compute* $\alpha_t = \min(\epsilon_0, \epsilon_0 \frac{\tau}{t})$ *and update*

$$\lambda^{(t+1)} = \lambda^{(t)} + \alpha_t \bar{g} / \sqrt{\bar{v}}$$

[6]The term $\nabla_\lambda \log q_\lambda(\theta_s) \circ \big(h_\lambda(\theta_s) - c\big)$ should be understood component-wise, i.e. it is the vector whose ith element is $\nabla_{\lambda_i} \log q_\lambda(\theta_s) \times \big(h_\lambda(\theta_s) - c_i\big)$.

– *Compute the lower bound estimate*

$$\widehat{LB}(\lambda^{(t)}) := \frac{1}{S}\sum_{s=1}^{S} h_{\lambda^{(t)}}(\theta_s).$$

– *If* $t \geq t_W$*: compute the moving averaged lower bound*

$$\overline{LB}_{t-t_W+1} = \frac{1}{t_W}\sum_{k=1}^{t_W} \widehat{LB}(\lambda^{(t-k+1)}),$$

and if $\overline{LB}_{t-t_W+1} \geq \max(\overline{\text{LB}})$ *patience = 0; else patience*:=*patience*+1.

– *If patience* $\geq P$, `stop=true`.

– *Set* $t := t+1$.

Example 3.1. With the model and data in Example 2.1, let's derive a FFVB procedure for approximating the posterior $p(\mu,\sigma^2|y) \propto p(\mu)p(\sigma^2)p(y|\mu,\sigma^2)$ using Algorithm 4. Suppose that the VB approximation is $q_\lambda(\mu,\sigma^2) = q(\mu)q(\sigma^2)$ with $q(\mu) = \mathcal{N}(\mu_\mu,\sigma_\mu^2)$ and $q(\sigma^2) = \text{Inverse-Gamma}(\alpha_{\sigma^2},\beta_{\sigma^2})$. This toy example is simply to demonstrate the use of Algorithm 4, we do not focus on the approximation accuracy here.

The model parameter is $\theta = (\mu,\sigma^2)^\top$ and the variational parameter $\lambda = (\mu_\mu,\sigma_\mu^2,\alpha_{\sigma^2},\beta_{\sigma^2})^\top$. In order to implement Algorithm 4, we need $h_\lambda(\theta) = h(\theta) - \log q_\lambda(\theta)$ with

$$\begin{aligned} h(\theta) &= \log\big(p(\mu)p(\sigma^2)p(y|\mu,\sigma^2)\big) \\ &= -\frac{n+1}{2}\log(2\pi) - \frac{1}{2}\log(\sigma_0^2) - \frac{(\mu-\mu_0)^2}{2\sigma_0^2} + \alpha_0\log(\beta_0) - \log\Gamma(\alpha_0) - (\frac{n}{2}+\alpha_0+1)\log(\sigma^2) \\ &\quad - \frac{\beta_0}{\sigma^2} - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i-\mu)^2, \end{aligned}$$

$$\log q_\lambda(\theta) = \alpha_{\sigma^2}\log\beta_{\sigma^2} - \log\Gamma(\alpha_{\sigma^2}) - (\alpha_{\sigma^2}+1)\log\sigma^2 - \frac{\beta_{\sigma^2}}{\sigma^2} - \frac{1}{2}\log(2\pi) - \frac{1}{2}\log(\sigma_\mu^2) - \frac{(\mu-\mu_\mu)^2}{2\sigma_\mu^2},$$

and

$$\nabla_\lambda \log q_\lambda(\theta) = \Big(\frac{\mu-\mu_\mu}{\sigma_\mu^2}, -\frac{1}{2\sigma_\mu^2} + \frac{(\mu-\mu_\mu)^2}{2\sigma_\mu^4}, \log\beta_{\sigma^2} - \frac{\Gamma'(\alpha_{\sigma^2})}{\Gamma(\alpha_{\sigma^2})} - \log\sigma^2, \frac{\alpha_{\sigma^2}}{\beta_{\sigma^2}} - \frac{1}{\sigma^2}\Big)^\top.$$

We are now ready to implement Algorithm 4. Figure 3 plots the estimate of the posterior densities together with the lower bound. The Variational Bayes estimates appear to be quite close to the Gibbs sampling estimates in this example, with some small discrepancy between them. These estimates can be improved with more advanced variants of FFVB presented later.

△

Algorithm 5 (FFVB with control variates and natural gradient)**. Input***: Initial* $\lambda^{(0)}$*, momentum weight* α_m*, fixed learning rate* ϵ_0*, threshold* τ*, rolling window size* t_W *and maximum patience* P*.* **Model-specific requirement***: function* $h(\theta) := \log\big(p(\theta)p(y|\theta)\big)$*.*

- *Initialization*

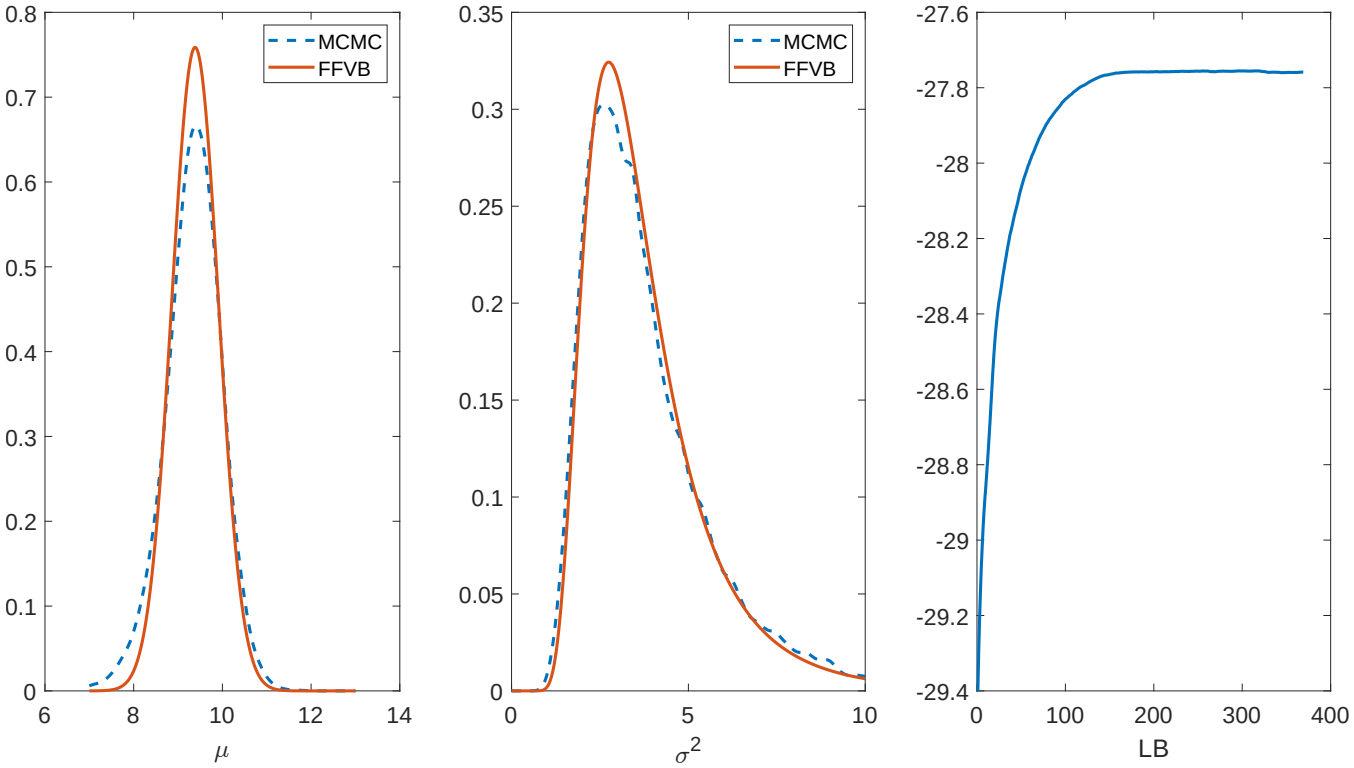

Figure 3: Example 3.1: Posterior density for μ and σ^2 estimated by FFVB Algorithm 4 and Gibbs sampling. The last panel shows the smoothened lower bounds $\overline{\text{LB}}_t$. The controlling parameters used are $S=2000$, $\beta_1=\beta_2=0.9$, $\epsilon_0=0.005$, $P=10$, $\tau=1000$ and $t_W=50$.

- *Generate* $\theta_s \sim q_{\lambda^{(0)}}(\theta)$, $s=1,...,S$.
- *Compute the unbiased estimate of the LB gradient*

$$\widehat{\nabla_\lambda LB}(\lambda^{(0)}) := \frac{1}{S}\sum_{s=1}^{S} \nabla_\lambda \log q_\lambda(\theta_s) \times h_\lambda(\theta_s)|_{\lambda=\lambda^{(0)}}$$

  and the natural gradient

$$\widehat{\nabla_\lambda \text{LB}}(\lambda^{(0)})^{nat} := I_F^{-1}(\lambda^{(0)})\widehat{\nabla_\lambda \text{LB}}(\lambda^{(0)}).$$

- *Set momentum gradient* $\overline{\nabla_\lambda \text{LB}} := \widehat{\nabla_\lambda \text{LB}}(\lambda^{(0)})^{nat}$.
- *Estimate control variate vector* c *as in* (22) *using the samples* $\{\theta_s, s=1,...,S\}$.
- *Set* $t=0$, *patience*$=0$ *and* `stop=false`.

- *While* `stop=false`:
  - *Generate* $\theta_s \sim q_{\lambda^{(t)}}(\theta)$, $s=1,...,S$.
  - *Compute the unbiased estimate of the LB gradient*

$$\widehat{\nabla_\lambda LB}(\lambda^{(t)}) = \frac{1}{S}\sum_{s=1}^{S} \nabla_\lambda \log q_\lambda(\theta_s) \circ \big(h_\lambda(\theta_s) - c\big)|_{\lambda=\lambda^{(t)}}$$

and the natural gradient

$$\widehat{\nabla_\lambda \mathrm{LB}}(\lambda^{(t)})^{nat} = I_F^{-1}(\lambda^{(t)})\widehat{\nabla_\lambda \mathrm{LB}}(\lambda^{(t)}).$$

- *Estimate the new control variate vector c as in (22) using the samples $\{\theta_s, s=1,...,S\}$.*
- *Compute the momentum gradient*

$$\overline{\nabla_\lambda \mathrm{LB}} = \alpha_m \overline{\nabla_\lambda \mathrm{LB}} + (1-\alpha_m)\widehat{\nabla_\lambda \mathrm{LB}}(\lambda^{(t)})^{nat}.$$

- *Compute $\alpha_t = \min(\epsilon_0, \epsilon_0 \frac{\tau}{t})$ and update*

$$\lambda^{(t+1)} = \lambda^{(t)} + \alpha_t \overline{\nabla_\lambda \mathrm{LB}}.$$

- *Compute the lower bound estimate*

$$\widehat{LB}(\lambda^{(t)}) := \frac{1}{S}\sum_{s=1}^{S} h_{\lambda^{(t)}}(\theta_s).$$

- *If $t \geq t_W$: compute the moving average lower bound*

$$\overline{LB}_{t-t_W+1} = \frac{1}{t_W}\sum_{k=1}^{t_W} \widehat{LB}(\lambda^{(t-k+1)}),$$

  and if $\overline{LB}_{t-t_W+1} \geq \max(\overline{\mathrm{LB}})$ patience = 0; else patience$:=$patience$+1$.
- *If patience$\geq P$,* `stop=true`*.*
- *Set $t := t+1$.*

Example 3.2. With the model and data in Example 2.1, let's derive a FFVB procedure for approximating the posterior $p(\mu,\sigma^2|y) \propto p(\mu)p(\sigma^2)p(y|\mu,\sigma^2)$ using Algorithm 5. In order to implement Algorithm 5, apart from $h_\lambda(\theta)$ and $\nabla_\lambda \log q_\lambda(\theta)$ as in Example 3.1, we need the Fisher information matrix I_F. It can be seen that this is a diagonal block matrix with two main blocks

$$\begin{pmatrix} \frac{1}{\sigma_\mu^2} & 0 \\ 0 & \frac{1}{2\sigma_\mu^4} \end{pmatrix}, \text{ and } \begin{pmatrix} \frac{\partial^2 \log\Gamma(\alpha_{\sigma^2})}{\partial \alpha_{\sigma^2} \partial \alpha_{\sigma^2}} & -\frac{1}{\beta_{\sigma^2}} \\ -\frac{1}{\beta_{\sigma^2}} & \frac{\alpha_{\sigma^2}}{\beta_{\sigma^2}^2} \end{pmatrix}.$$

Figure 4 shows the estimated densities together with the lower bound estimates. In this example, Algorithm 5 appears to produce a very similar approximation as in Algorithm 4.

△

The choice of the variational distribution $q_\lambda(\mu,\sigma^2) = q(\mu)q(\sigma^2)$ in Examples 3.1 and 3.2 ignores the posterior dependence between μ and σ^2. There are several alternatives that can improve this. One of these is to use Gaussian VB (see Section 3.5) to approximate the posterior of the transformed parameter $\theta = \big(\mu, \log(\sigma^2)\big)$. Another alternative is presented in Example 3.3 below.

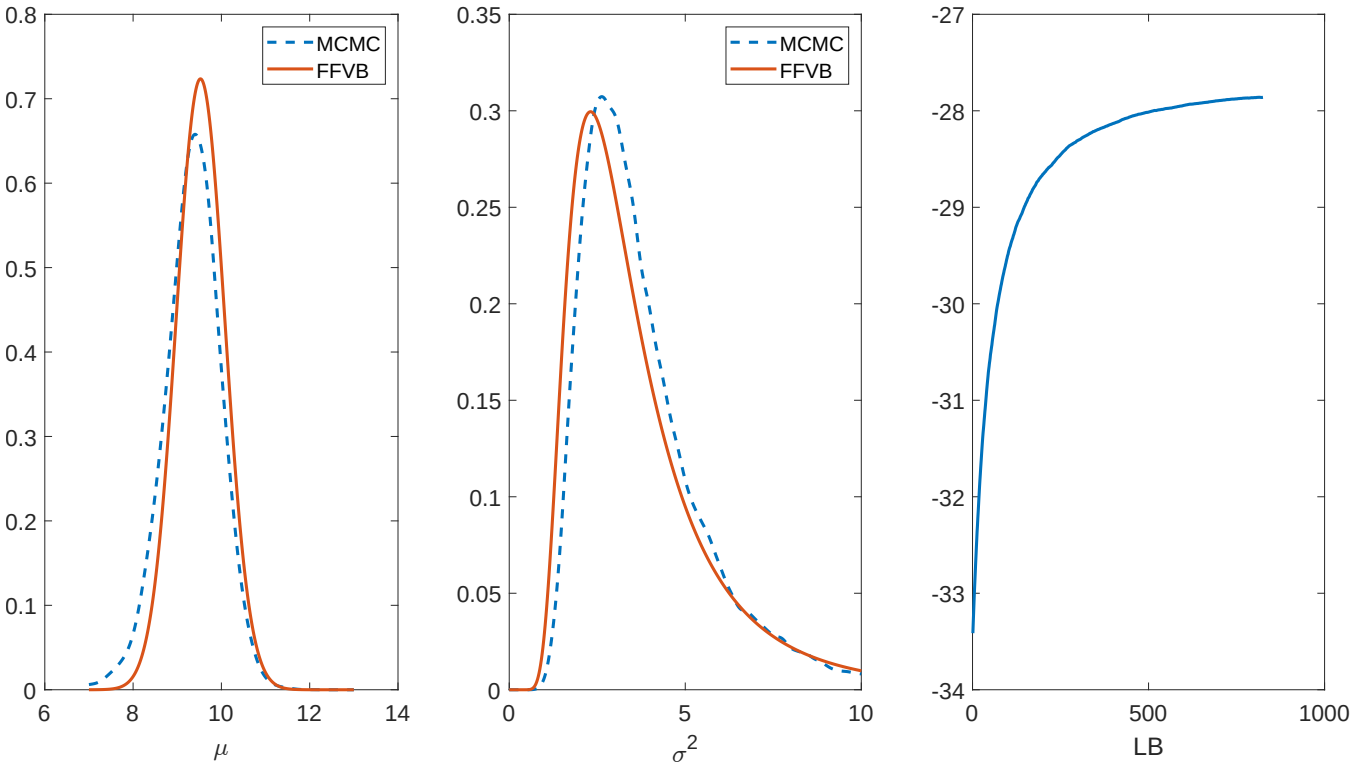

Figure 4: Example 3.2: Posterior density for μ and σ^2 estimated by FFVB Algorithm 5 and Gibbs sampling. The last panel shows the averaged lower bounds $\overline{\mathrm{LB}}_t$.

Example 3.3. Consider again the model and data in Example 2.1. It is possible to exploit the structure of this model to develop a better VB approximation. Let us derive a FFVB procedure for approximating the posterior $p(\mu,\sigma^2|y) \propto p(\mu)p(\sigma^2)p(y|\mu,\sigma^2)$ using the variational distribution with density of the form

$$q_\lambda(\mu,\sigma^2) = \widetilde{q}_\lambda(\mu)p(\sigma^2|y,\mu), \quad \widetilde{q}_\lambda(\mu) = \mathcal{N}(\mu_\mu,\sigma_\mu^2). \tag{23}$$

This distribution, as the joint distribution of μ and σ^2, doesn't have a standard form, however, it is straightforward to sample from it. This variational distribution exploits the standard form of the full conditional $p(\sigma^2|y,\mu)$, which is inverse-Gamma, and takes into account the posterior dependence between μ and σ^2.

The variational parameter λ now only consists of μ_μ and σ_μ^2. Using (16), the gradient of the lower bound is

$$\nabla_\lambda \mathrm{LB}(\lambda) = \mathbb{E}_{q_\lambda(\mu,\sigma^2)}\Big(\nabla_\lambda \log \widetilde{q}_\lambda(\mu) \times h_\lambda(\theta)\Big)$$

with

$$h_\lambda(\theta) = \log p(\mu,\sigma^2) + \log p(y|\mu,\sigma^2) - \log \widetilde{q}_\lambda(\mu) - \log p(\sigma^2|y,\mu).$$

Algorithm 4 or Algorithm 5 now can be applied.

Figure 5 shows the estimated results. As shown, this "hybrid" VB approximation is highly accurate in terms of both marginal density estimate and the joint density estimate.

△

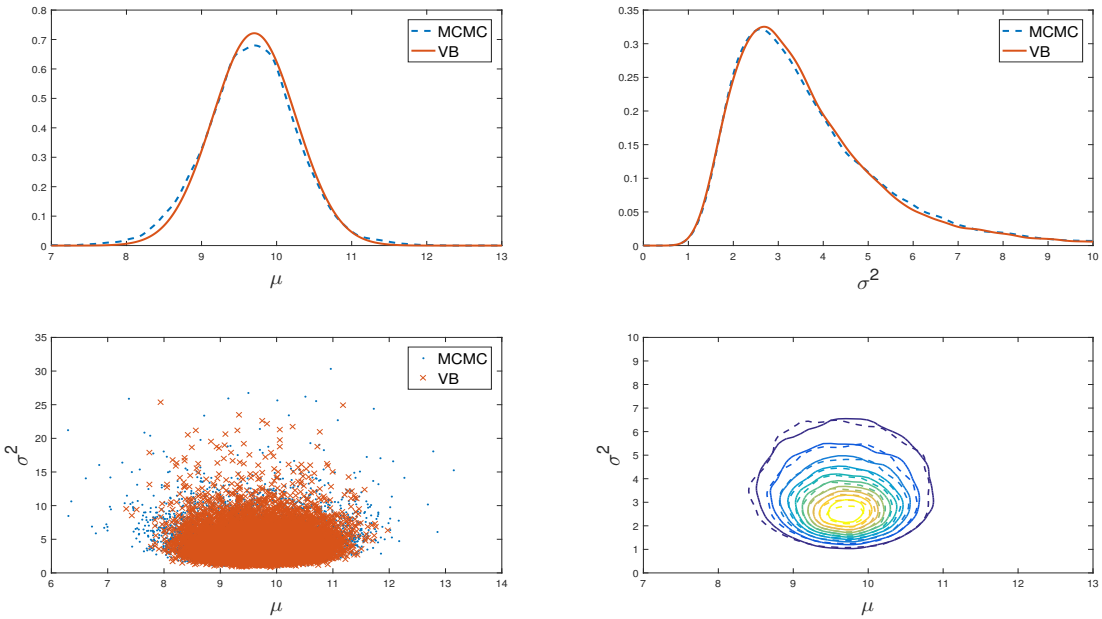

Figure 5: Example 3.3: First row: Posterior densities for μ and σ^2 estimated by Gibbs sampling and the hybrid VB method in (23). Second row: The joint samples and contour plot for the estimated joint posterior. In the bottom-right corner plot, the dashed lines are contours estimated based on the Gibbs samples, and the solid lines estimated based on the samples generated from (23).

3.4 Reparameterization trick

The reparameterization trick is an attractive alternative to the control variate in Section 3.3. Suppose that for $\theta \sim q_\lambda(\cdot)$, there exists a deterministic function $g(\lambda,\varepsilon)$ such that $\theta = g(\lambda,\varepsilon) \sim q_\lambda(\cdot)$ where $\varepsilon \sim p_\varepsilon(\cdot)$. We emphasize that $p_\varepsilon(\cdot)$ must not depend on λ. For example, if $q_\lambda(\theta) = \mathcal{N}(\theta;\mu,\sigma^2)$ then $\theta = \mu + \sigma\varepsilon$ with $\varepsilon \sim \mathcal{N}(0,1)$. Writing $\mathrm{LB}(\lambda)$ as an expectation with respect to $p_\varepsilon(\cdot)$

$$\mathrm{LB}(\lambda) = \mathbb{E}_{\varepsilon\sim p_\varepsilon}\Big(h_\lambda(g(\varepsilon,\lambda))\Big),$$

where $\mathbb{E}_{\varepsilon\sim p_\varepsilon}(\cdot)$ denotes expectation with respect to $p_\varepsilon(\cdot)$, and differentiating under the integral sign gives

$$\nabla_\lambda \mathrm{LB}(\lambda) = \mathbb{E}_{\varepsilon\sim p_\varepsilon}\Big(\nabla_\lambda g(\lambda,\varepsilon)^\top \nabla_\theta h_\lambda(\theta)\Big) + \mathbb{E}_{\varepsilon\sim p_\varepsilon}\Big(\nabla_\lambda h_\lambda(\theta)\Big)$$

where the θ within $h_\lambda(\theta)$ is understood as $\theta = g(\varepsilon,\lambda)$ with λ fixed. In particular, the gradient $\nabla_\lambda h_\lambda(\theta)$ is taken when θ is not considered as a function of λ. Here, with some abuse of notation, $\nabla_\lambda g(\lambda,\varepsilon)$ denotes the Jacobian matrix of size $d_\theta \times d_\lambda$ of the vector-valued function $\theta = g(\lambda,\varepsilon)$. Note that

$$\begin{aligned}\mathbb{E}_{\varepsilon\sim p_\varepsilon}\Big(\nabla_\lambda h_\lambda(\theta)\Big) = \mathbb{E}_{\varepsilon\sim q_\varepsilon}\Big(\nabla_\lambda h_\lambda\big(\theta = g(\varepsilon,\lambda)\big)\Big) &= -\mathbb{E}_{\varepsilon\sim q_\varepsilon}\Big(\nabla_\lambda \log q_\lambda\big(\theta = g(\varepsilon,\lambda)\big)\Big) \\ &= -\mathbb{E}_{\theta\sim q_\lambda}\Big(\nabla_\lambda \log q_\lambda(\theta)\Big) = 0,\end{aligned}$$

hence

$$\nabla_\lambda \mathrm{LB}(\lambda) = \mathbb{E}_{\varepsilon\sim q_\varepsilon}\Big(\nabla_\lambda g(\lambda,\varepsilon)^\top \nabla_\theta h_\lambda(\theta)\Big). \tag{24}$$

The gradient (24) can be estimated unbiasedly using i.i.d samples $\varepsilon_s \sim p_\varepsilon(\cdot)$, $s=1,...,S$, as

$$\widehat{\nabla_\lambda \mathrm{LB}}(\lambda) = \frac{1}{S}\sum_{s=1}^{S} \nabla_\lambda g(\lambda,\varepsilon_s)^\top \nabla_\theta \big\{h_\lambda(g(\lambda,\varepsilon_s))\big\}. \tag{25}$$

The *reparametrization gradient* estimator (25) is often more efficient than alternative approaches to estimating the lower bound gradient, partly because it takes into account the information from the gradient $\nabla_\theta h_\lambda(\theta)$. In typical VB applications, the number of Monte Carlo samples S used in estimating the lower bound gradient can be as small as 5 if the reparameterization trick is used, while the control variates method requires an S of about hundreds or more. However, there is a dilemma about choosing S that we must be careful of. With the reparameterization trick, a small S might be enough for estimating the lower bound gradient, however, we still need a moderate S in order to obtain a good estimate of the lower bound if lower bound is used in the stopping criterion. Also, compared to score-function gradient, FFVB approaches that use reparameterization gradient require not only the model-specific function $h(\theta)$ but also its gradient $\nabla_\theta h(\theta)$.

Algorithm 6 provides a detailed implementation of the FFVB approach that uses the reparameterization trick and adaptive learning. A small modification of Algorithm 6 (not presented) gives the implementation of the FFVB approach that uses the reparameterization trick and natural gradient.

Algorithm 6 (FFVB with reparameterization trick and adaptive learning)**. Input***: Initial $\lambda^{(0)}$, adaptive learning weights $\beta_1,\beta_2 \in (0,1)$, fixed learning rate ϵ_0, threshold τ, rolling window size t_W and maximum patience P.* **Model-specific requirement***: function $h(\theta) := \log\big(p(\theta)p(y|\theta)\big)$ and its gradient $\nabla_\theta h(\theta)$.*

- *Initialization*
  - *Generate $\varepsilon_s \sim p_\varepsilon(\cdot)$, $s=1,...,S$.*
  - *Compute the unbiased estimate of the LB gradient*

$$\widehat{\nabla_\lambda LB}(\lambda^{(0)}) := \frac{1}{S}\sum_{s=1}^{S} \nabla_\lambda g(\lambda,\varepsilon_s)^\top \nabla_\theta \big\{h_\lambda(g(\lambda,\varepsilon_s))\big\}\big|_{\lambda=\lambda^{(0)}}.$$

  - *Set $g_0 := \widehat{\nabla_\lambda LB}(\lambda^{(0)})$, $v_0 := (g_0)^2$, $\bar{g} := g_0$, $\bar{v} := v_0$.*
  - *Set $t=0$, patience$=0$ and* `stop=false`*.*
- *While* `stop=false`*:*
  - *Generate $\varepsilon_s \sim p_\varepsilon(\cdot)$, $s=1,...,S$*
  - *Compute the unbiased estimate of the LB gradient*

$$g_t := \widehat{\nabla_\lambda LB}(\lambda^{(t)}) = \frac{1}{S}\sum_{s=1}^{S} \nabla_\lambda g(\lambda,\varepsilon_s)^\top \nabla_\theta \big\{h_\lambda(g(\lambda,\varepsilon_s))\big\}\big|_{\lambda=\lambda^{(t)}}.$$

– *Compute* $v_t = (g_t)^2$ *and*

$$\bar{g} = \beta_1 \bar{g} + (1-\beta_1) g_t,\ \bar{v} = \beta_2 \bar{v} + (1-\beta_2) v_t.$$

– *Compute* $\alpha_t = \min(\varepsilon_0, \varepsilon_0 \frac{\tau}{t})$ *and update*

$$\lambda^{(t+1)} = \lambda^{(t)} + \alpha_t \bar{g} / \sqrt{\bar{v}}$$

– *Compute the lower bound estimate*

$$\widehat{LB}(\lambda^{(t)}) := \frac{1}{S} \sum_{s=1}^{S} h_{\lambda^{(t)}}(\theta_s),\ \theta_s = g(\lambda^{(t)}, \varepsilon_s).$$

– *If* $t \geq t_W$*: compute the moving average lower bound*

$$\overline{LB}_{t-t_W+1} = \frac{1}{t_W} \sum_{k=1}^{t_W} \widehat{LB}(\lambda^{(t-k+1)}),$$

and if $\overline{LB}_{t-t_W+1} \geq \max(\overline{\mathrm{LB}})$ *patience = 0; else patience*$:=$*patience*$+1$.

– *If patience* $\geq P$, `stop=true`.

– *Set* $t := t+1$.

3.5 Gaussian Variational Bayes

The most popular VB approaches are probably Gaussian VB where the approximation $q_\lambda(\theta)$ is a Gaussian distribution with mean μ and covariance matrix Σ. This section presents several variants of this GVB approach.

3.5.1 GVB with Cholesky decomposed covariance

This GVB method uses the Cholesky decomposition for the covariance matrix Σ, $\Sigma = LL^\top$ with L a lower triangular matrix[7]. We will use the reparameterization trick for variance reduction. A sample $\theta \sim q_\lambda(\theta)$ can be written as $\theta = g(\lambda, \varepsilon) = \mu + L\varepsilon$ with $\varepsilon \sim \mathcal{N}_d(0, I_d)$, and d the dimension of θ. The variational parameter vector λ includes μ and the non-zero elements of L. As Jacobian matrix $\nabla_\mu g(\lambda, \varepsilon) = I$, the identity matrix, from (24), the gradient of the lower bound w.r.t. μ is

$$\nabla_\mu \mathrm{LB}(\lambda) = \mathbb{E}_\varepsilon \big[\nabla_\theta h_\lambda(\theta) \big], \quad \text{with} \quad \theta = \mu + L\varepsilon.$$

To compute the gradient w.r.t. L, we first need some notations. For a $d \times d$ matrix A, denote by $\mathrm{vec}(A)$ the d^2-vector obtained by stacking the columns of A from left to right one underneath the other, by $\mathrm{vech}(A)$ the $\frac{1}{2}d(d+1)$-vector obtained by stacking the columns of the lower triangular part of A, and by $A \otimes B$ the Kronecker product of matrices A and B. For any matrices A, B and X

[7] For the Cholesky decomposition of Σ to be unique, one needs the constraint that the diagonal entries of L to be strictly positive. For simplicity, however, we do not impose this constraint here.

of suitable sizes, we shall use the fact that $\mathrm{vec}(AXB)=(B^\top\otimes A)\mathrm{vec}(X)$. Then, $L\varepsilon=\mathrm{vec}(I_dL\varepsilon)=(\varepsilon^\top\otimes I_d)\mathrm{vec}(L)$ and hence $\nabla_{\mathrm{vec}(L)}g(\lambda,\varepsilon)=\varepsilon^\top\otimes I_d$. From (24),

$$\begin{aligned}\nabla_{\mathrm{vec}(L)}\mathrm{LB}(\lambda) &= \mathbb{E}_\varepsilon\Big[\nabla_{\mathrm{vec}(L)}g(\lambda,\varepsilon)^\top\nabla_\theta h_\lambda(\theta)\Big]\\ &= \mathbb{E}_\varepsilon\Big[(\varepsilon\otimes I_d)\nabla_\theta h_\lambda(\theta)\Big]\\ &= \mathbb{E}_\varepsilon\Big[\mathrm{vec}\big(\nabla_\theta h_\lambda(\theta)\varepsilon^\top\big)\Big],\quad \text{with}\quad \theta=\mu+L\varepsilon.\end{aligned}$$

This implies that

$$\nabla_{\mathrm{vech}(L)}\mathrm{LB}(\lambda)=\mathbb{E}_\varepsilon\big[\mathrm{vech}\big(\nabla_\theta h_\lambda(\theta)\varepsilon^\top\big)\big]. \tag{26}$$

From Algorithm 6, we arrive at the following GVB algorithm, referred to below as Cholesky GVB.

Algorithm 7 (Cholesky GVB)**. Input**: *Initial* $\mu^{(0)}$, $L^{(0)}$ *and* $\lambda^{(0)}:=(\mu^{(0)\top},\mathrm{vech}(L^{(0)})^\top)^\top$, *number of samples* S, *adaptive learning weights* $\beta_1,\beta_2\in(0,1)$, *fixed learning rate* ϵ_0, *threshold* τ, *rolling window size* t_W *and maximum patience* P. **Model-specific requirement**: *function* $h(\theta)$ *and* $\nabla_\theta h(\theta)$.

- *Initialization*
  - *Generate* $\varepsilon_s\sim N_d(0,I)$, $s=1,...,S$.
  - *Compute the estimate of the lower bound gradient* $\widehat{\nabla}_\lambda\mathrm{LB}(\lambda^{(0)})=(\widehat{\nabla}_\mu\mathrm{LB}(\lambda^{(0)})^\top,\widehat{\nabla}_{\mathrm{vech}(L)}\mathrm{LB}(\lambda^{(0)})^\top)^\top$ *where*

$$\begin{aligned}\widehat{\nabla}_\mu\mathrm{LB}(\lambda^{(0)}) &:= \frac{1}{S}\sum_{s=1}^S\nabla_\theta h_\lambda(\theta_s),\\ \widehat{\nabla}_{\mathrm{vech}(L)}\mathrm{LB}(\lambda^{(0)}) &:= \frac{1}{S}\sum_{s=1}^S\mathrm{vech}\big(\nabla_\theta h_\lambda(\theta_s)\varepsilon_s^\top\big),\end{aligned}$$

    with $\theta_s=\mu^{(0)}+L^{(0)}\varepsilon_s$.
  - *Set* $g_0:=\widehat{\nabla}_\lambda\mathcal{L}(\lambda^{(0)})$, $v_0:=(g_0)^2$, $\bar{g}:=g_0$, $\bar{v}:=v_0$.
  - *Set* $t=0$, *patience*$=0$ *and* `stop=false`.
- *While* `stop=false`*:*
  - *Generate* $\varepsilon_s\sim p_\varepsilon(\cdot)$, $s=1,...,S$. *Recalculate* $\mu^{(t)}$ *and* $L^{(t)}$ *from* $\lambda^{(t)}$.
  - *Compute the estimate of the lower bound gradient* $g_t:=\widehat{\nabla}_\lambda\mathrm{LB}(\lambda^{(t)})=(\widehat{\nabla}_\mu\mathrm{LB}(\lambda^{(t)})^\top,\widehat{\nabla}_{\mathrm{vech}(L)}\mathrm{LB}(\lambda^{(t)})^\top)^\top$ *where*

$$\begin{aligned}\widehat{\nabla}_\mu\mathrm{LB}(\lambda^{(t)}) &:= \frac{1}{S}\sum_{s=1}^S\nabla_\theta h_\lambda(\theta_s),\\ \widehat{\nabla}_{\mathrm{vech}(L)}\mathrm{LB}(\lambda^{(t)}) &:= \frac{1}{S}\sum_{s=1}^S\mathrm{vech}\big(\nabla_\theta h_\lambda(\theta_s)\varepsilon_s^\top\big),\end{aligned}$$

    with $\theta_s=\mu^{(t)}+L^{(t)}\varepsilon_s$.

– *Compute* $v_t = (g_t)^2$ *and*

$$\bar{g} = \beta_1 \bar{g} + (1-\beta_1) g_t,\ \bar{v} = \beta_2 \bar{v} + (1-\beta_2) v_t.$$

– *Compute* $\alpha_t = \min(\varepsilon_0, \varepsilon_0 \frac{\tau}{t})$ *and update*

$$\lambda^{(t+1)} = \lambda^{(t)} + \alpha_t \bar{g} / \sqrt{\bar{v}}$$

– *Compute the lower bound estimate*

$$\widehat{\mathcal{L}}(\lambda^{(t)}) := \frac{1}{S} \sum_{s=1}^{S} h_\lambda(\theta_s).$$

– *If* $t \geq t_W$*: compute the moving averaged lower bound*

$$\overline{\mathcal{L}}_{t-t_W+1} = \frac{1}{t_W} \sum_{k=1}^{t_W} \widehat{\mathcal{L}}(\lambda^{(t-k+1)}),$$

and if $\overline{\mathcal{L}}_{t-t_W+1} \geq \max(\overline{\mathrm{LB}})$ *patience = 0; else patience*:= *patience*+1.

– *If patience* $\geq P$, `stop=true`.

– *Set* $t := t+1$.

Example 3.4 (Bayesian logistic regression)**.** Consider a Bayesian logistic regression problem with design matrix $X = [x_1, ..., x_n]^\top$ and vector of binary responses y. The log-likelihood is

$$\log p(y|X,\theta) = y^\top X\theta - \sum_{i=1}^{n} \log\big(1 + \exp(x_i^\top \theta)\big)$$

with θ the vector of d coefficients. Suppose that a normal prior $\mathcal{N}(0, \sigma_0^2 I)$ is used for θ. To implement the Cholesky GVB method, all we need is the function

$$h(\theta) = \log p(\theta) + \log p(y|X,\theta) = -\frac{d}{2}\log(2\pi) - \frac{d}{2}\log(\sigma_0^2) - \frac{\theta^\top \theta}{2\sigma_0^2} + y^\top X\theta - \sum_{i=1}^{n} \log\big(1 + \exp(x_i^\top \theta)\big), \quad (27)$$

and its gradient

$$\nabla_\theta h(\theta) = -\frac{1}{\sigma_0^2}\theta + X^\top \big(y - \pi(\theta)\big) \quad (28)$$

with

$$\pi(\theta) = \Big(\frac{1}{1+\exp(-x_1^\top \theta)}, \cdots, \frac{1}{1+\exp(-x_n^\top \theta)}\Big)^\top. \quad (29)$$

The Labour Force Participation dataset contains information of 753 women with one binary variable indicating whether or not they are currently in the labour force together with seven covariates such as number of children under 6 years old, age, education level, etc. Figure 6 plots the VB approximation for each coefficient θ_i together with the lower bound estimates over the iterations.

△

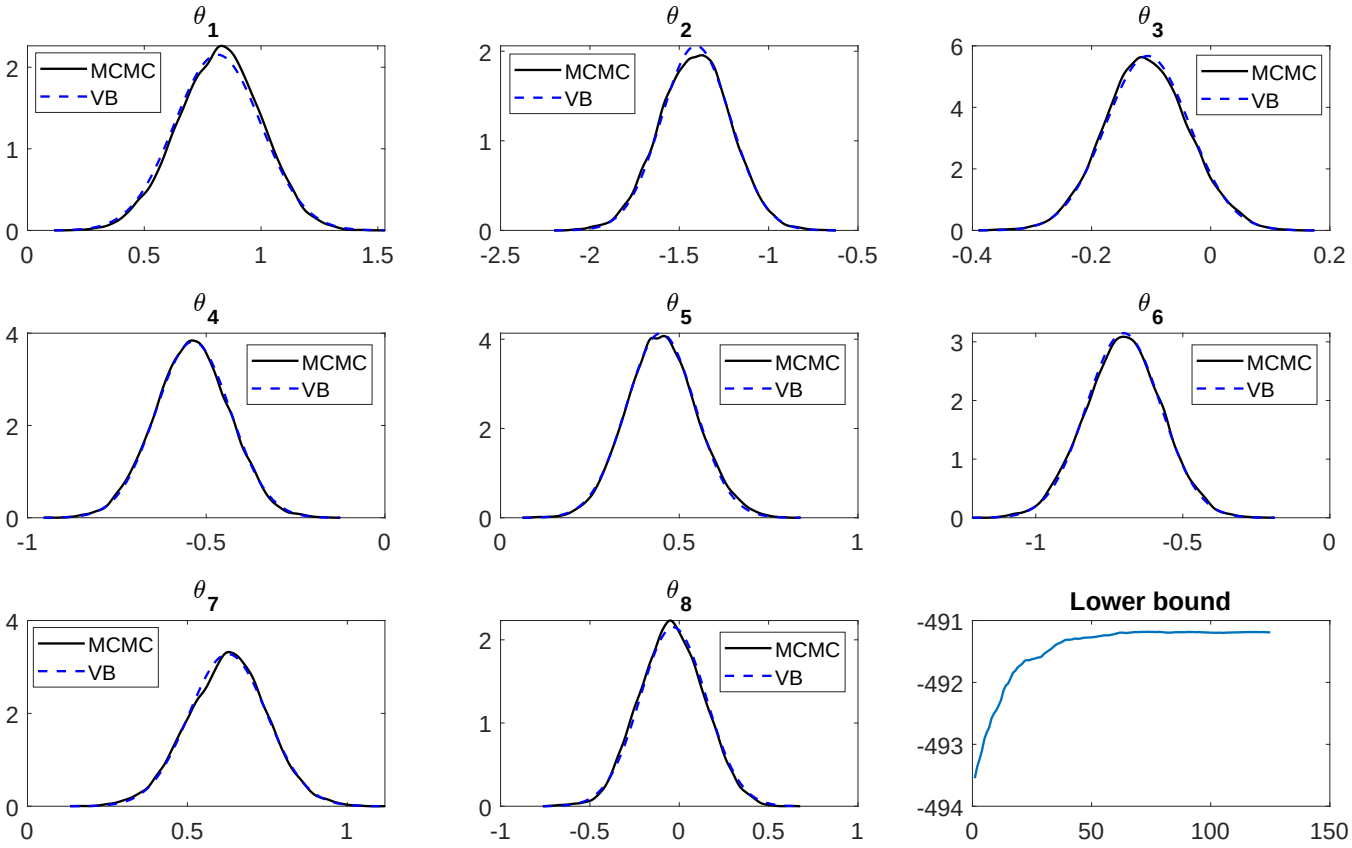

Figure 6: Cholesky GVB for approximating the posterior in logistic regression. The CPU time was roughly 3 seconds. The controlling parameters used are $S=50$, $\beta_1=\beta_2=0.9$, $\epsilon_0=0.002$, $P=20$, $\tau=500$ and $t_W=50$.

3.5.2 GVB with factor decomposed covariance

An alternative to the Cholesky decomposition is the factor decomposition

$$\Sigma = BB^\top + C^2,$$

where B is the factor loading matrix of size $d\times f$ with $f \ll d$ the number of factors and C is a diagonal matrix, $C=\text{diag}(c_1,...,c_d)$. GVB with this factor covariance structure is useful in high-dimensional settings where d is large, as the number of variational parameters reduces from $d+d*(d+1)/2$ in the case of full Gaussian to $(f+2)d$ in the case of factor decomposition. This VB method is first developed in Ong et al. (2018) who term the method Variational Approximation with Factor Covariance (VAFC) and use Algorithm 6 for training, as computing the natural gradient in this case is difficult. This section describes the case with one factor, $f=1$, which achieves a great computational speed-up for approximate Bayesian inference in big models such as deep neural networks where d can be very large. Also, with $f=1$, Tran et al. (2020b) show that it is possible to calculate the natural gradient efficiently and term their method NAtural gradient Gaussian Variational Approximation with factor Covariance (NAGVAC).

With $f=1$, we rewrite the factor decomposition as

$$\Sigma = bb^\top + C^2, \quad C=\text{diag}(c),$$

where $b=(b_1,...,b_d)^\top$ and $c=(c_1,...,c_d)^\top$ are vectors. The variational parameter vector is $\lambda=(\mu^\top,b^\top,c^\top)^\top$. Using the reparameterization trick, $\theta\sim\mathcal{N}(\mu,\Sigma)$ can be written as

$$\theta = g(\lambda,\varepsilon) = \mu + \varepsilon_1 b + c\circ\varepsilon_2$$

where $\varepsilon=(\varepsilon_1,\varepsilon_2^\top)^\top\sim\mathcal{N}_{d+1}(0,I)$, and $c\circ\varepsilon_2$ denotes the component-wise product of vectors c and ε_2. Note that

$$\nabla_\mu g(\lambda,\varepsilon)=I_d,\ \nabla_b g(\lambda,\varepsilon)=\varepsilon_1 I_d,\ \nabla_c g(\lambda,\varepsilon)=\text{diag}(\varepsilon_2),$$

hence the reparameterization gradient is

$$\nabla_\lambda \text{LB}(\lambda) = \mathbb{E}_{q_\varepsilon}\begin{pmatrix}\nabla_\theta h_\lambda(\mu+\varepsilon_1 b+c\circ\varepsilon_2)\\ \varepsilon_1\nabla_\theta h_\lambda(\mu+\varepsilon_1 b+c\circ\varepsilon_2)\\ \varepsilon_2\circ\nabla_\theta h_\lambda(\mu+\varepsilon_1 b+c\circ\varepsilon_2)\end{pmatrix}. \tag{30}$$

The gradient of function $h_\lambda(\theta)$ is

$$\nabla_\theta h_\lambda(\theta)=\nabla_\theta h(\theta)-\nabla_\theta\log q_\lambda(\theta)=\nabla_\theta\log\big(p(\theta)p(y|\theta)\big)-\nabla_\theta\log q_\lambda(\theta),$$

where the first term is model-specific and the second term is $\nabla_\theta\log q_\lambda(\theta)=-\Sigma^{-1}(\theta-\mu)$. To avoid computing directly the inverse Σ^{-1} and the matrix-vector multiplication, noting that $\Sigma^{-1}=C^{-2}-\frac{1}{1+b^\top C^{-2}b}C^{-2}bb^\top C^{-2}$, we have

$$\nabla_\theta \log q_\lambda(\theta) = -(\theta-\mu)\circ c^{-2} + \frac{(b\circ c^{-2})^\top(\theta-\mu)}{1+(b\circ c^{-1})^\top(b\circ c^{-1})}(b\circ c^{-2}),$$

with $c^{-1}:=(1/c_1,...,1/c_d)^\top$ and $c^{-2}:=(1/c_1^2,...,1/c_d^2)^\top$. To compute lower bound estimates, we need

$$\log q_\lambda(\theta) = -\frac{d}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma| - \frac{1}{2}(\theta-\mu)^\top\Sigma^{-1}(\theta-\mu).$$

As $\Sigma=C\big((C^{-1}b)(C^{-1}b)^\top+I\big)C$,

$$|\Sigma| = |C|^2\big(1+(C^{-1}b)^\top(C^{-1}b)\big) = \big(\prod_{i=1}^d c_i^2\big)\big(1+\sum_{i=1}^d\frac{b_i^2}{c_i^2}\big).$$

Hence, a computationally efficient version of $\log q_\lambda(\theta)$ is

$$\begin{aligned}\log q_\lambda(\theta) &= -\frac{d}{2}\log(2\pi) - \frac{1}{2}\sum_{i=1}^d\log c_i^2 - \frac{1}{2}\log\big(1+\sum_{i=1}^d\frac{b_i^2}{c_i^2}\big)\\ &\quad - \frac{1}{2}(\theta-\mu)^\top\big((\theta-\mu)\circ c^{-2}\big) + \frac{\big((b\circ c^{-2})^\top(\theta-\mu)\big)^2}{2\big(1+(b\circ c^{-1})^\top(b\circ c^{-1})\big)}.\end{aligned}$$

Finally, it can be shown that the natural gradient in (19) can be approximately computed in closed form as in the following algorithm (see Tran et al. (2020b)), whose computational complexity is $O(d)$.

Algorithm 8 (Computing the natural gradient)**.** *Input: Vector b, c and ordinary gradient of the lower bound $g=(g_1^\top,g_2^\top,g_3^\top)^\top$ with g_1 the vector formed by the first d elements of g, g_2 formed by the next d elements, and g_3 the last d elements. Output: The natural gradient $g^{nat}=I_F^{-1}g$.*

- *Compute the vectors $v_1=c^2-2b^2\circ c^{-4}$, $v_2=b^2\circ c^{-3}$, and the scalars $\kappa_1=\sum_{i=1}^d b_i^2/c_i^2$, $\kappa_2=\frac{1}{2}(1+\sum_{i=1}^d v_{2i}^2/v_{1i})^{-1}$.*
- *Compute*

$$g^{nat}=\begin{pmatrix}(g_1^\top b)b+c^2\circ g_1\\ \frac{1+\kappa_1}{2\kappa_1}\Big((g_2^\top b)b+c^2\circ g_2\Big)\\ \frac{1}{2}v_1^{-1}\circ g_3+\kappa_2\big[(v_1^{-1}\circ v_2)^\top g_3\big](v_1^{-1}\circ v_2)\end{pmatrix}.$$

We now describe the NAGVAC algorithm that can be used as a fast VB method for approximate Bayesian inference in high-dimensional applications such as Bayesian deep neural networks. In such applications, instead of using the lower bounds for stopping rule, one often uses a loss function evaluated on a validation dataset for stopping. Then, the updating is stopped if the loss function is not decreased after P iterations.

Algorithm 9 (NAGVAC)**. Input***: Initial $\lambda^{(0)}:=(\mu^{(0)},b^{(0)},c^{(0)})$, number of samples S, momentum weight α_m, fixed learning rate ϵ_0, threshold τ, rolling window size t_W and maximum patience P.* **Model-specific requirement***: function $h(\theta)$ and $\nabla_\theta h(\theta)$.*

- *Initialization*
  - *Generate $\varepsilon_{1,s}\sim\mathcal{N}(0,1)$ and $\varepsilon_{2,s}\sim\mathcal{N}_d(0,I_d)$, $s=1,...,S$.*
  - *Compute the lower bound gradient estimate $\widehat{\nabla}_\lambda \mathrm{LB}(\lambda^{(0)})$ as in (30), and then compute the natural gradient $\widehat{\nabla_\lambda \mathrm{LB}}(\lambda^{(0)})^{nat}$ using Algorithm 8.*
  - *Set momentum gradient $\overline{\nabla_\lambda \mathrm{LB}}:=\widehat{\nabla_\lambda \mathrm{LB}}(\lambda^{(0)})^{nat}$.*
  - *Set $t=0$, patience$=0$ and* `stop=false`*.*
- *While* `stop=false`*:*
  - *Generate $\varepsilon_{1,s}\sim\mathcal{N}(0,1)$ and $\varepsilon_{2,s}\sim\mathcal{N}_d(0,I_d)$, $s=1,...,S$.*
  - *Compute the lower bound gradient estimate $\widehat{\nabla}_\lambda \mathrm{LB}(\lambda^{(t)})$ as in (30), and then compute the natural gradient $\widehat{\nabla_\lambda \mathrm{LB}}(\lambda^{(t)})^{nat}$ using Algorithm 8.*
  - *Compute the momentum gradient*

$$\overline{\nabla_\lambda \mathrm{LB}}=\alpha_m\overline{\nabla_\lambda \mathrm{LB}}+(1-\alpha_m)\widehat{\nabla_\lambda \mathrm{LB}}(\lambda^{(t)})^{nat}.$$

  - *Compute $\alpha_t=\min(\epsilon_0,\epsilon_0\frac{\tau}{t})$ and update*

$$\lambda^{(t+1)}=\lambda^{(t)}+\alpha_t\overline{\nabla_\lambda \mathrm{LB}}.$$

  - *Compute the validation loss $Loss(\lambda^{(t)})$. If $Loss(\lambda^{(t)})\le\min\{Loss(\lambda^{(1)}),...,Loss(\lambda^{(t-1)})\}$ patience = 0; else patience$:=$patience$+1$.*

– *If patience* $\geq P$, `stop=true`.

– *Set* $t := t+1$.

Example 3.5 (Bayesian deep neural net)**.** This example briefly presents an application of the NAGVAC method for fitting a Bayesian deep neural network (BNN). See Tran et al. (2020b) for a detailed description of this example. Bayesian deep neural network models are an example of big models where the size d of unknown parameters can be in thousands or millions. We consider the census dataset extracted from the U.S. Census Bureau database and available on the UCI Machine Learning Repository `https://archive.ics.uci.edu/ml/index.php`. The prediction task is to determine whether a person's income is over \$50K per year, based on 14 attributes including age, workclass, race, etc, of which many are categorical variables. After using dummy variables to represent the categorical variables, there are 103 input variables. The training dataset has 24,129 observations and the validation set has 6032 observations. As is typical in Deep Learning applications, here we use the minus log-likelihood computed on the validation set as the loss function to judge when to stop the VB training algorithm. The structure of the neural net is [104,100,100]: input layer with 104 variables including the intercept, and two hidden layers each with 100 units. The size of parameters θ is 20,500. Algorithm 9 for training this deep learning model stopped after 2812 iterations. Figure 7 plots the validation loss over the iterations. For a detailed discussion on the prediction accuracy of this BNN compared to the Bayesian logistic model, see Tran et al. (2020b).

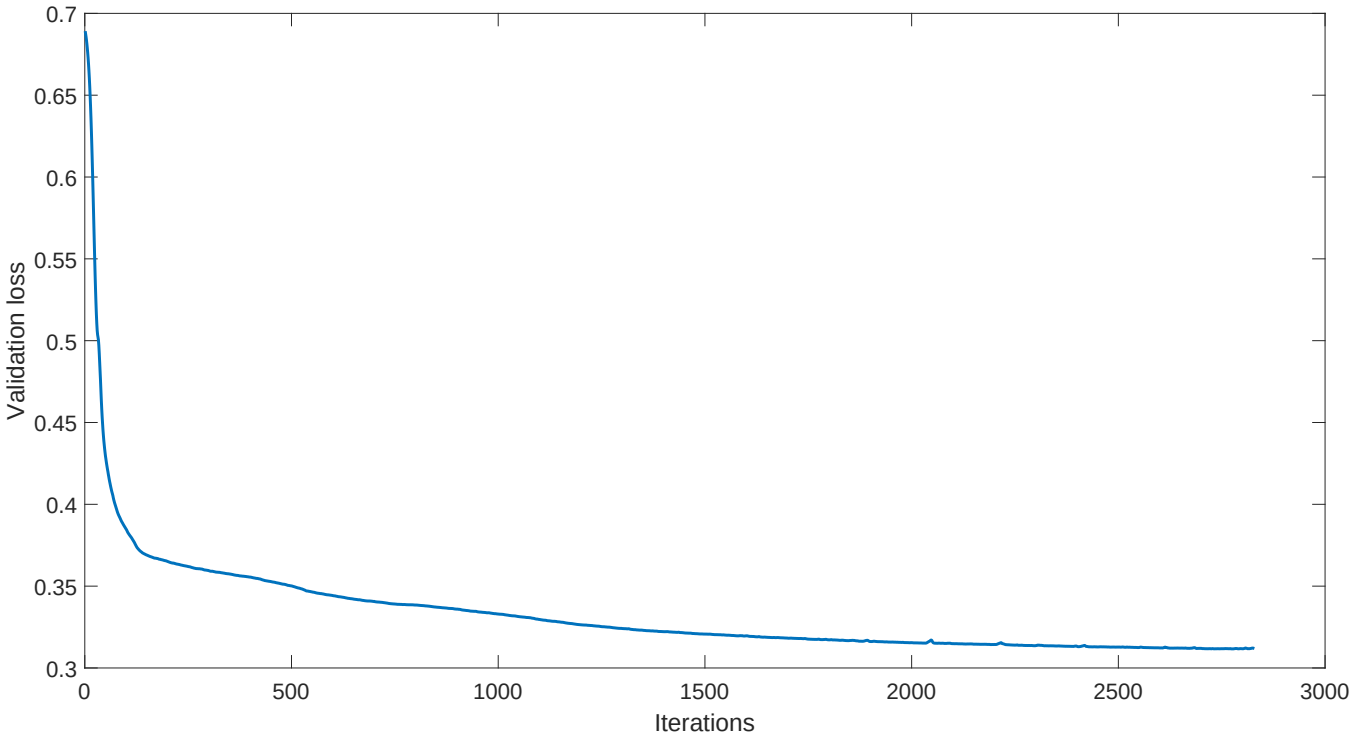

Figure 7: Example 3.5: The NAGVAC method in Algorithm 9 for deep neural net modelling. The plot shows the validation loss over iterations.

△

3.6 Practical recommendation

We conclude this section with a few practical recommendations that have been found useful in practice. First, the fixed learning rate ϵ_0 in (18) requires some effort to tune, often based on trial and error. Good starting values are $\epsilon_0 = 0.01$ or $\epsilon_0 = 0.001$, then adjusted after a few runs. If the VB algorithm converges too quickly, it is probably because ϵ_0 is set too large and needs to be reduced. Plotting the moving averaged lower bounds is a convenient and useful way for implementation diagnostic. If this plot fluctuates too much, then a larger number of samples S is needed, and also a wider moving average window t_W should be used. If these moving averaged lower bounds show a clear trend of decreasing, then something must have gone wrong.

It is often useful to standardize the data before model fitting. For example, in regression modelling, each numerical column in the input matrix X should be standardized to have mean zero and standard deviation of 1.

For challenging applications, it is a good idea to run the FFVB algorithm several times with different initialization $\lambda^{(0)}$ and select the one that ends up with the largest final lower bound. It is also common to fix the random seed so that the results are reproducible.

Finally, a simple practice known as *gradient clipping* is often found useful. Gradient clipping makes the gradient estimate more well behaved by clipping its length while still maintaining its direction. It replaces the lower bound gradient estimate $\widehat{\nabla_\lambda \mathrm{LB}}(\lambda)$ by

$$\frac{\ell_{\text{threshold}}}{\|\widehat{\nabla_\lambda \mathrm{LB}}(\lambda)\|} \widehat{\nabla_\lambda \mathrm{LB}}(\lambda), \tag{31}$$

if the ℓ_2-norm $\|\widehat{\nabla_\lambda \mathrm{LB}}(\lambda)\|$ is larger than some threshold $\ell_{\text{threshold}}$, such as 100 or 1000. Note that we used gradient clipping in Examples 3.4 and 3.5.

3.7 A quick note on the bibliography of FFVB

This isn't a review paper, we therefore made no attempt to give a comprehensive literature review on Variational Bayes. In addition to Section 2.2, this section is to give a short list of further reading on FFVB for the interested reader. Compared to MFVB, FFVB is developed more recently with great contributions not only from machine learning but also the statistics community. Further reading on control variate can be found in Paisley et al. (2012); Nott et al. (2012); Ranganath et al. (2014); Tran et al. (2017). See Kingma and Ba (2014); Duchi et al. (2011) and Zeiler (2012) for the adaptive learning methods. The natural gradient is first introduced in statistics, in the context of MLE, by Rao (1945), popularized in machine learning by Amari (1998), and developed further for applications in Variational Bayes by Sato (2001); Hoffman et al. (2013); Martens (2014); Khan and Lin (2017); Lin et al. (2019) and Tran et al. (2020b). The reparameterization trick can be found in Kingma and Welling (2014); Titsias and Lázaro-Gredilla (2014). The Cholesky GVB in Section 3.5.1 is borrowed from Titsias and Lázaro-Gredilla (2014) and Tan and Nott (2018), and more details of Algorithm 8 together with the deep learning model in Example 3.5 can be found in Tran et al. (2020b).

There are more advanced variants of FFVB, such as the Importance Weighted Lower Bound of Burda et al. (2016) and manifold VB of Tran et al. (2020a), that aren't presented in this tutorial. Also, there are recent advances in theoretical properties of VB approximations that we don't cover here; the interested reader is referred to Alquier and Ridgway (2020) and Zhang and Gao (2020).

4 VBLab software package and its applications

This section describes our end-user software package, VBLab, that implements several general FFVB algorithms described in Section 3, and demonstrates their use. The package also implements several other FFVB algorithms, such as the VAFC of Ong et al. (2018) and manifold VB of Tran et al. (2020a), that are not described in Section 3.

VBLab is a probabilistic programming software package, currently available in Matlab, allowing automatic variational Bayesian inference on many pre-defined common statistical models and also user-defined models. The package provides various FFVB methods and works efficiently for high dimensional and complex posterior distributions. Users are not required to know the technicality behind the VB techniques provided; all they need to do is to supply their statistical model, which can be specified flexibly in various ways.

4.1 Bayesian logistic regression

We consider again the Bayesian logistic regression model in Example 3.4 and demonstrate how to use the VBLab package to output a VB approximation of the posterior distribution using the Cholesky GVB.

First, the Labour Force Participation dataset is loaded by calling the `readData()` function with `'LabourForce'` string as its input argument:

```
% Load the Labour Force Participation dataset
labour = readData('LabourForce',...
                  'Intercept',true);  % Add column of 1 as intercept
```

This dataset, together to several others, are included in the package and can be loaded using the `readData()` function of the VBLab package. For the purpose of this example, we use the entire Labour Force Participation data to train the model; if necessary, users can split the data into a training and testing data using the `trainTestSplit()` function.

Next, we create a logistic regression model object which is an instance of the `LogisticRegression` class as follows:

```
% Number of parameters of the Logistic Regression model
n_features = size(labour,2)-1;

% Create a Logistic Regression model object
Mdl = LogisticRegression(n_features,...
                         'Prior',{'Normal',[0,50]});
```

The `LogisticRegression` model class requires at least one input argument indicating the number of model parameters. The optional argument `'Prior'` sets the prior for each coefficient of the regression model; here, a normal prior with zero mean and variance 50 is used. By default, `'Prior'` is set to be the standard normal distribution. The VBLab package provides commonly-used prior distributions including Normal, Uniform, Beta, Exponential, Gamma, Inverse-Gamma, Binomial and many others.

Given the logistic regression model object `Mdl`, we now can call any FFVB algorithm provided in the VBLab package to produce a variational approximation of the posterior distribution. The following code calls the Cholesky GVB algorithm class `CGVB`:

```
% Run Cholesky GVB
Post_CGVB = CGVB(Mdl,labour,...
                 'LearningRate',0.002,...  % Learning rate
                 'NumSample',50,...        % Number of VB samples
                 'MaxPatience',20,...      % For Early stopping
                 'MaxIter',5000,...        % Maximum number of iterations
                 'InitMethod','Custom',... % Randomly initialize variational mean
                 'GradWeight1',0.9,...     % Momentum weight 1
                 'GradWeight2',0.9,...     % Momentum weight 2
                 'WindowSize',50,...       % Smoothing window for lowerbound
                 'GradientMax',10,...      % For gradient clipping
                 'LBPlot',true);           % Plot the lowerbound when finish
```

The algorithm class `CGVB` requires several input arguments specifying how this VB algorithm is implemented. The first argument is the statistical model of interest `Mdl`, which can be defined as a class object or a function handle. The second argument is the dataset `labour`, which can be either a Matlab table, or a single matrix with the last column to be the response data y. Table 2 lists all the optional arguments of the `CGVB` class together with their equivalent mathematical notations and default values.

Argument	Default value	Notation	Description
`LearningRate`	0.002	ϵ_0	Fixed learning rate in (18)
`NumSample`	50	S	Monte Carlo samples
`MaxPatience`	20	P	Maximum patience
`GradWeight1`	0.9	β_1	Adaptive learning weight
`GradWeight2`	0.9	β_2	Adaptive learning weight
`WindowSize`	50	t_W	Rolling window size
`StepAdaptive`	`MaxIter`/2	τ	Threshold to start reducing learning rates
`MaxIter`	1000		Maximum number of iterations
`GradientMax`	10	$\ell_{\text{threshold}}$	Gradient clipping threshold in (31)
`InitMethod`	Random		Initialization method
`LBPlot`	true		Whether or not to plot the lower bounds

Table 2: Input arguments of the `CGVB` class constructor with their equivalent mathematical notations and default values.

The outputs of the CGVB algorithm class are store in the attribute `Post`, which is a Matlab structure data type, of the output `Post_CGVB`. Table 3 lists the fields of the `Post` attribute together with their descriptions and equivalent notations in Section 3.5.1. For example, the following code shows how to extract the mean μ and variance $\text{diag}(\Sigma)$ of the Gaussian variational distribution, and then plots the corresponding normal density using `vbayesPlot()` function of the VBLab package, as shown in Figure 8:

```
% Extract variational mean and variance
mu_vb = Post_CGVB.Post.mu;           % Varational mean
sigma2_vb = Post_CGVB.Post.sigma2;   % Variational variance
```

Output	Description	Notation
`LB`	The lower bound estimated in each iteration	$\widehat{\text{LB}}(\lambda)$
`LB_smooth`	The smoothed lower bound estimated in each iteration	$\overline{\text{LB}}(\lambda)$
`mu`	Mean of the Gaussian variational distribution	μ
`L`	The lower triangular matrix of the variational covariance matrix	L
`Sigma`	The variational covariance matrix	Σ
`sigma2`	The diagonal of the variational covariance matrix	$\text{diag}(\Sigma)$

Table 3: Outputs of the CGVB algorithm together with their description and equivalent mathematical notations.

```
% Plot the variational distribution of each model parameter
for i=1:num_feature
    subplot(3,3,i)
    vbayesPlot('Density',...
               'Distribution',{'Normal',[mu_vb(i),sigma2_vb(i)]})
end
```

We can also extract the smoothed lower bounds from `Post` and plot them as shown in the last panel of Figure 8:

```
% Plot the smoothed lower bound
subplot(3,3,9)
plot(Post_CGVB.Post.LB_smooth)
title('Lower bound')
```

4.2 Bayesian deep neural networks

This section demonstrates how to use the VBLab package for variational Bayesian inference in Bayesian deep neural networks. The package implements the DeepGLM model of Tran et al. (2020b), which provides a unified framework for flexible regression that combines the deep neural network method in machine learning for data representation with the popular Generalized Linear Models (GLM) in statistics. This section also describes the use of the `VAFC` class that implements the GVB with factor covariance (VAFC) algorithm briefly mentioned in Section 3.5.2.

We first load the German Credit data by calling the `readData()` function with `'GermanCredit'` string as its input argument:

```
% Load the German Credit dataset
credit = readData('GermanCredit',...
                  'Type','Table',...     % Store data in a table
                  'Intercept',true,...   % Add column of 1 for intercept
                  'Normalized',true);    % Neural Networks work more efficient
                                         % with normalized data

% Number of input features
n_features = size(credit,2) - 1;
```

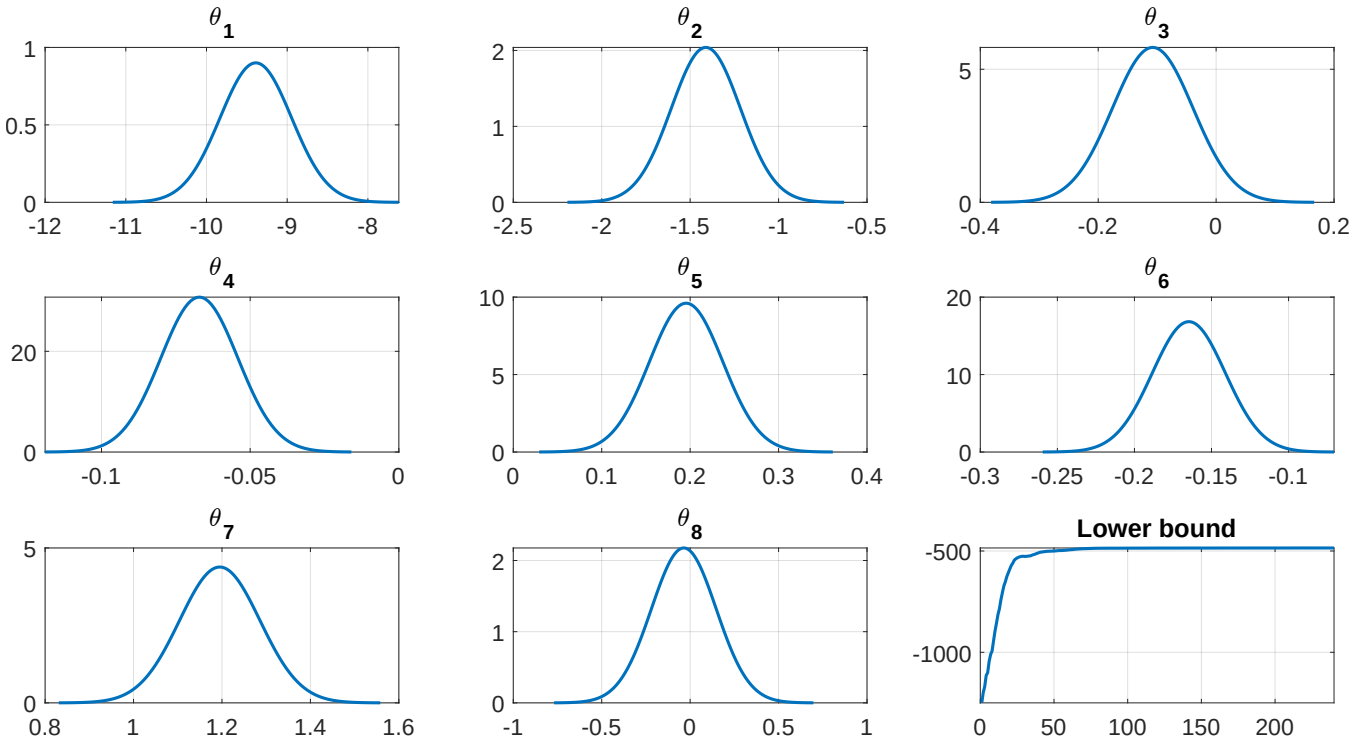

Figure 8: Variational distribution densities of model parameters and the smoothed lower bound.

We then define an instance of the `DeepGLM` model class, which specifies important components such as the prior, likelihood function, etc., for the DeepGLM model:

```
% Define a deepGLM model object
Mdl = DeepGLM([n_features,10,10],...
              'Activation','Relu',...
              'Distribution','Binomial');
```

The `DeepGLM` model class requires at least one input argument which is the structure of the neural network. The code above specifies a structure that has one input layer with `n_features` units (including the bias term), and two hidden layers each with 10 units. The `'Activation'` argument, set to `'Relu'` by default, specifies the activation function used for each hidden unit. The `'Distribution'` argument, is `'Normal'` by default, specifies the distribution used for the response data. In the code above, we set `'Distribution'` to be `'Binomial'` as the response variable in the German Credit data is binary. Users are referred to the documentation of the VBLab package for a more comprehensive discussion on the `DeepGLM` class.

Finally, we run the `VAFC` algorithm class to approximate the posterior distribution of this DeepGLM model (for bigger DeepGLM models or if computational speed-up is of primary importance, one should use the `NAGVAC` algorithm class rather than `VAFC`):

```
% Run VAFC to obtain VB approximation of the posterior
Post_VAFC = VAFC(Mdl,credit,...
                 'Validation',0.2,...
                 'LearningRate',0.002,...
                 'NumFactor',4,...
```

```
                'NumSample',50,...
                'GradWeight',0.9,...
                'MaxPatience',100,...
                'MaxIter',10000,...
                'GradientMax',200,...
                'WindowSize',30,...
                'InitMethod','Random');
```

The `'NumFactor'` argument, 4 in this example, specifies the number of factors used in VAFC. As we use a prediction loss on a validation dataset to assess the convergence of the VAFC algorithm, we split `data` into a training set, for parameter estimation, and a validation set, for early stopping. The `'Validation'` argument, set to be 0.2 in this example, indicates that we use 20% of the data to form the validation set.

4.3 Volatility modelling with the RECH models

Let $y=\{y_t, t=1,...,T\}$ be a time series of financial asset returns and $\mathcal{F}_t$ be the σ-field of the information up to time t. Volatility, defined as the conditional variance $\sigma_t^2 := \text{Var}(y_t|\mathcal{F}_{t-1})$, is of high interest in the financial sector. Conditional heteroskedastic models, such as GARCH of Bollerslev (1986), represent σ_t^2 as a deterministic function of the observations and conditional variances in the previous time steps. Nguyen et al. (2020) recently propose a new class of conditional heteroskedastic models, namely the REcurrent Conditional Heteroskedastic (RECH) models, by combining recurrent neural networks (RNNs) and GARCH-type models, for flexible modelling of the volatility dynamics. The conditional variance in the RECH models is the sum of two components: the recurrent component modeled by an RNN, and the garch component modeled by a GARCH-type structure. For example, by using the Simple Recurrent Network (SRN) for the recurrent component ω_t and the standard GARCH(1,1) for the garch component, they obtain the SRN-GARCH specification of the RECH models as:

$$y_t = \sigma_t \epsilon_t,\ \epsilon_t \overset{iid}{\sim} \mathcal{N}(0,1),\ t=1,2,...,T \tag{32a}$$
$$\sigma_t^2 = \omega_t + \alpha y_{t-1}^2 + \beta \sigma_{t-1}^2,\ t=2,...,T,\ \sigma_1^2 = \sigma_0^2 \tag{32b}$$
$$\omega_t = \beta_0 + \beta_1 h_t,\ t=2,...,T, \tag{32c}$$
$$h_t = \phi(v x_t + w h_{t-1} + b),\ t=2,...,T, \text{ with } h_1 \equiv 0; \tag{32d}$$

Nguyen et al. (2020) suggest $x_t = (\omega_{t-1}, y_{t-1}, \sigma_{t-1}^2)^\top$. The SRN-GARCH model has 7 parameters: $\theta = (\alpha, \beta, \beta_0, \beta_1, v, w, b)$.

The following code shows how to use the VBLab package for Bayesian inference in RECH using the Manifold GVB method of Tran et al. (2020a). First, we read the SP500 data by calling the `readData()` function

```
% Load the SP500 daily return data
sp500 = readData('RealizedLibrary',...
                 'Index','SP500',...
                 'Length',1000);        % Extract only the last 1000 observations
```

In this example, we use only the last 1000 observations to perform the approximation Bayesian inference by setting the value of the `'Length'` argument to be 1000.

Next, we define a RECH model together with its prior:

```
% Define priors for model parameters using 2D cell array
% Parameter names must be specified correctly
prior = {{'v','w','b'},'Normal',[0,1];...
         {'beta0','beta1'},'Inverse-Gamma',[0.25,2.5];...
         {'alpha','beta'},'Uniform',[0,1]};

% Define a RECH model with SRN-GARCH specification
Mdl = RECH('SRN-GARCH',...
           'Prior', prior);
```

We define the priors for the SRN-GARCH's parameters using a Matlab 2D cell array. Each row of this cell array has three elements including: parameter names, name of the prior distribution and its parameters. The parameter names are put in a 1D cell array listing the model parameters that share the same prior distribution. The prior distribution name must be one of the distribution classes available in the VBLab package. The distribution parameters must be stored in a Matlab 1D array. In this example, we use the same priors as suggested in Nguyen et al. (2020). The model class `RECH` requires at least one input argument, which is a particular specification of the RECH models. The current version of the VBLab package provides three specifications for the RECH models including `'SRN-GARCH'`, `'SRN-GRJ'` and `'SRN-EGARCH'`. The `'Prior'` argument sets the priors for model parameters defined previously in the variable `prior`. Users can refer to the documentation of the VBLab package for more comprehensive discussion on the `RECH` model class.

Given the model object `rech_model` defined by the `RECH` model class, we now run the Manifold GVB method by calling the `MGVB` algorithm class:

```
% Run MGVB given the data and RECH model
Post_MGVB = MGVB(Mdl,y,...
                 'NumSample',100,...
                 'LearningRate',0.01,...
                 'GradWeight',0.4,...
                 'MaxPatience',50,...
                 'MaxIter',2500,...
                 'GradientMax',100,...
                 'WindowSize',30);
```

Similar to the other VB algorithm classes, the `MGVB` class stores the outputs in a Matlab structure which can be used as shown in the following code to visualize the density of variational distribution and smoothed lower bound.

```
% Extract variation mean and variance
mu_vb     = Post_MGVB.Post.mu;
sigma2_vb = Post_MGVB.Post.sigma2;

% Define parameter names for plotting
param_name = {'\beta_0','\beta_1','\alpha','\beta','v','w','b'};

% Plot the variational distribution of each parameter
for i=1:num_feature
    subplot(3,3,i)
    vbayesPlot('Density',...
               'Distribution',{'Normal',[mu_vb(i),sigma2_vb(i)]})
    title(param_name{i})
end
```

```
% Plot the smoothed lower bound
subplot(3,3,9)
plot(Post_MGVB.Post.LB_smooth)
title('Lower bound')
```

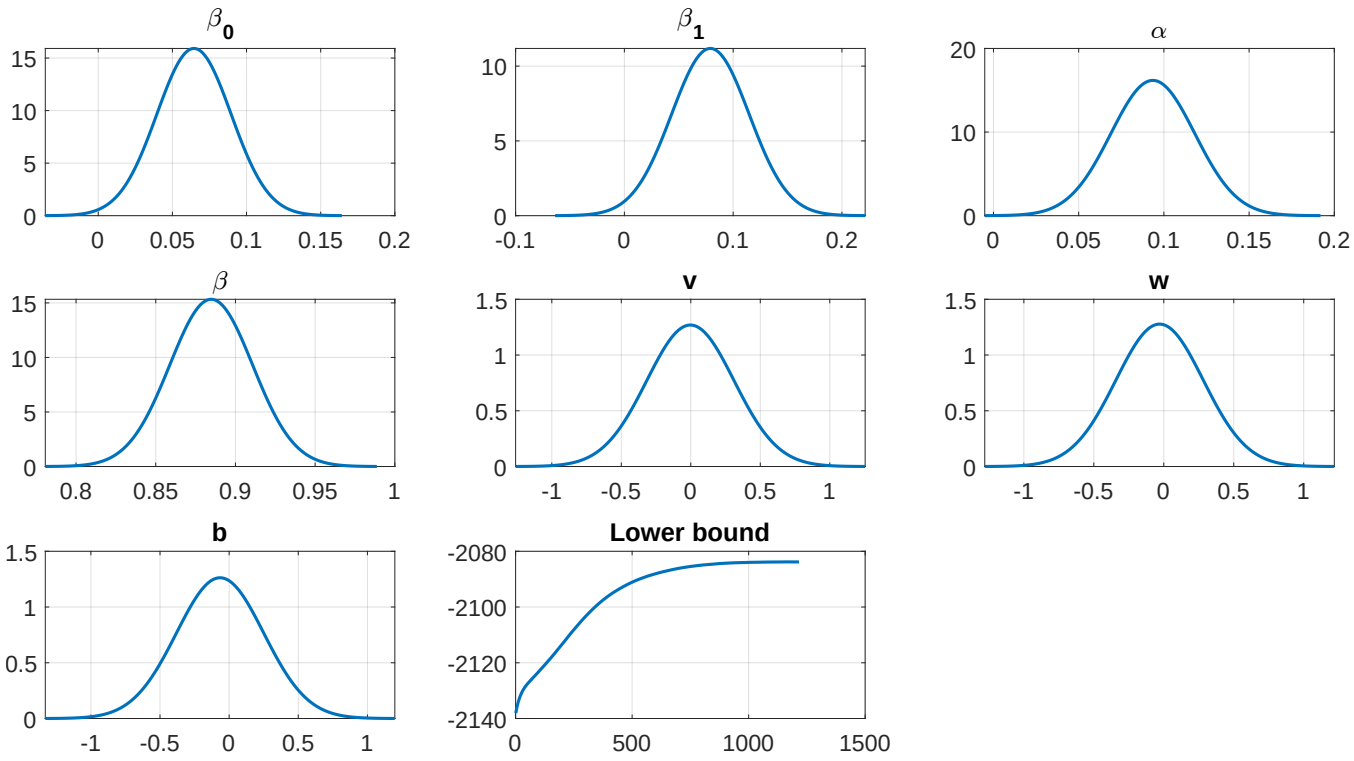

Figure 9: Variational distribution densities of the RECH model parameters and the smoothed lower bound.

4.4 Using the VBLab package for user-defined models

For pre-defined models such as logistic regression or DeepGLM, we can use the model classes provided in the VBLab package to create the corresponding model object before calling a VB algorithm as demonstrated in the previous sections. For user-defined statistical models, the package provides several ways for custom-built models that work with VB algorithm classes such as `CGVB`, `VAFC`, `MGVB` and `NAGVAC`. It only requires users to specify a function to compute $h(\theta)$ and $\nabla_\theta h(\theta)$ as discussed in Algorithm 7 and 9. We demonstrate this use of the package below using logistic regression.

4.4.1 Bayesian logistic regression with manual gradient

Users need to supply a function that evaluates the model-specific term $h(\theta)$. For VB algorithms that are based on the reparameterization trick, users also need to supply the gradient $\nabla_\theta h(\theta)$. This section considers the case where $\nabla_\theta h(\theta)$ can be calculated manually, and Section 4.4.2 demonstrates how to use Automatic Differentiation to calculate this gradient.

The following code defines a function that computes both $h(\theta)$ and $\nabla_\theta h(\theta)$ as in (27)-(28):

```
function [h_func_grad,h_func] = grad_h_func_logistic(data,theta,mdl)

    % Extract additional settings
    d = length(theta);
    sigma2 = mdl.Prior(2);

    % Extract data
    X = data(:,1:end-1);
    y = data(:,end));

    % Compute log likelihood
    aux = X*theta;
    llh = y.*aux-log(1+exp(aux));
    llh = sum(llh);

    % Compute gradient of log likelihood
    ppi       = 1./(1+exp(-aux));
    llh_grad  = X'*(y-ppi);

    % Compute log prior
    log_prior =-d/2*log(2*pi)-d/2*log(sigma2)-theta'*theta/sigma2/2;

    % Compute gradient of log prior
    log_prior_grad = -theta/sigma2;

    % Compute h(theta) = log p(y|theta) + log p(theta)
    h_func = llh + log_prior;

    % Compute gradient of the h(theta)
    h_func_grad = llh_grad + log_prior_grad;

    % h_func_grad must be a column
    h_func_grad = reshape(h_func_grad,length(h_func_grad),1);

end
```

There are some rules to define a proper function for calculating $h(\theta)$ and $\nabla h(\theta)$ that it is compatible with the VB algorithm classes in the package.

- The input should have three arguments:
  - `data`: The data that is used for calculating $h(\theta)$ and $\nabla_\theta h(\theta)$.
  - `theta`: A *column* vector of model parameters.
  - `mdl`: Any additional setting necessary for defining the custom-built models. This `mdl` variable must be created before running VB algorithms and used as the input to the `'Setting'` argument of the VB algorithm classes.
- There are two outputs:
  - The first output, e.g. `h_func_grad` as in the previous code, must be a *column* vector that returns the value of $\nabla_\theta h(\theta)$.

– The second output, e.g. `h_func` as in the previous code, must be a *scalar* that returns the value $h(\theta)$.

To assist with calculating gradient, VBLab provides the static methods `logPdfFnc()` and `GradlogPdfFnc()` for conveniently computing the log density and its gradient for common prior distributions. For example, rather than having to specify log normal density and its gradient explicitly as in the code above, one can call the functions `Normal.logPdfFnc(theta,mu,sigma2)` and `Normal.GradlogPdfFnc(theta,mu` to compute the log density and its gradient, respectively, of the Gaussian distribution with mean `mu` and variance `sigma2`.

Given the function to compute $h(\theta)$ and $\nabla h(\theta)$, we now use a VB algorithm class, e.g. `CGVB`, to produce a variational approximation of the posterior distribution defined by $h(\theta)$:

```
% Load the Labour Force Participation dataset
labour = readData('LabourForce',...
                  'Type','Matrix',...
                  'Intercept',true);

% Number of model parameters. Adding 1 for the intercept.
n_features = size(labour,2)-1;

% Struct to store prior
setting.Prior = [0,50];

% Initialize the variational mean
mu_init = normrnd(0,0.01,n_features,1);

% Create an CGVB object and run the CGVB algorirthm
Post_CGVB = CGVB(@grad_h_func_logistic,labour,...
                 'NumParams',n_features,...
                 'Setting',setting,...
                 'MeanInit',mu_init,...
                 'LearningRate',0.002,...
                 'NumSample',50,...
                 'MaxPatience',20,...
                 'MaxIter',5000,...
                 'GradWeight1',0.9,...
                 'GradWeight2',0.9,...
                 'WindowSize',50,...
                 'GradientMax',10,...
                 'LBPlot',true);
```

After loading the data, we create the structure `setting` to store additional variables, rather than the data and model parameters, necessary for computing $h(\theta)$ and $\nabla_\theta h(\theta)$. The handle of the user-defined function `grad_h_func_logistic` is passed to the `CGVB` class constructor as the first input argument. We also need to set the value of the argument `'NumParams'` to be the number of model parameters and pass the variable `setting` to the `'Setting'` argument. The `CGVB` class provides several ways to initialize the variational mean μ. In this example, we initialize μ randomly using a normal distribution, which is used as the input to the `'MeanInit'` argument. The other algorithmic arguments of the `CGVB` class are set as in Section 4.1.

4.4.2 Bayesian logistic regression with Automatic Differentiation

Instead of computing the gradient $\nabla_\theta h(\theta)$ manually as in the previous section, we can compute it using Matlab's Automatic Differentiation facility, which is a technique for evaluating derivatives numerically and automatically. The general rule of using Automatic Differentiation in Matlab is that we must call `dlgradient()` inside a helper function, and then evaluate the gradient using `dlfeval()`. The following code modifies the function `grad_h_func_logistic()` above to output $\nabla_\theta h(\theta)$ using Automatic Differentiation.

```
% Define a function to compute h(theta) = log p(theta) + log p(y|theta)
function h_func = h_func_logistic(data,theta,mdl)

    % Extract additional settings
    d = length(theta);
    sigma2 = mdl.Prior(2);

    % Extract data
    X = data(:,1:end-1);
    y = data(:,end));

    % Compute log likelihood
    aux = X*theta;
    log_lik = y.*aux-log(1+exp(aux));
    log_lik = sum(log_lik);

    % Compute log prior
    log_prior =-d/2*log(2*pi)-d/2*log(sigma2)-theta'*theta/sigma2/2;

    % h = log p(y|theta) + log p(theta)
    h_func = llh + log_prior;

end


% Define a function to call dlgradient to automatically compute the gradient
% of the h function
function [h_func_grad,h_func] = grad_h_func_logistic_AD(data,theta,mdl)

    h_func = h_func_logistic(data,theta,mdl);
    h_func_grad = dlgradient(h_func,theta);

end


function [h_func_grad,h_func] = grad_h_func_logistic(data,theta,mdl)

    % Convert parameters to dlarray data type
    theta_AD = dlarray(theta);

    % Evaluate the function containing dlgradient using dlfeval
    [h_func_grad_AD,h_func_AD] = ...
       dlfeval(@grad_h_func_logistic_AD,data,theta_AD,mdl);

    % Convert parameters from dlarray to matlab array
```

```
    h_func_grad = extractdata(h_func_grad_AD);
    h_func = extractdata(h_func_AD);

    % Make sure the output is a column vector
    h_func_grad = reshape(h_func_grad,length(h_func_grad),1);

end
```

This `grad_h_func_logistic` now can be used as the first input argument of the `CGVB` algorithm class as before.

References

Alquier, P. and Ridgway, J. (2020). Concentration of tempered posteriors and of their variational approximations. *Annals of Statistics*, 48(3):1475–1497.

Amari, S. (1998). Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. New York: Springer.

Bollerslev, T. (1986). Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 31(3):307 – 327.

Burda, Y., Grosse, R., and Salakhutdinov, R. (2016). Importance weighted autoencoders. *Proceedings of the 4th International Conference on Learning Representations (ICLR)*.

Corduneanu, A. and Bishop, C. (2001). Variational Bayesian model selection for mixture distributions. In Jaakkola, T. and Richardson, T., editors, *Artifcial Intelligence and Statistics*, volume 14, pages 27–34. Morgan Kaufmann.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

Ghahramani, Z. and Hinton, G. E. (2000). Variational learning for switching state-space models. *Neural computation*, 12(4):831–864.

Giordani, P., Mun, X., Tran, M.-N., and Kohn, R. (2013). Flexible multivariate density estimation with marginal adaptation. *Journal of Computational and Graphical Statistics*, 22(4):814–829.

Hoffman, M. D., Blei, D. M., Wang, C., and Paisley, J. (2013). Stochastic variational inference. *Journal of Machine Learning Research*, 14:1303–1347.

Jordan, M., Ghahramani, Z., Jaakkola, T., and Saul, L. K. (1999). An introduction to variational methods for graphical models. *Machine Learning*, 37:183–233.

Khan, M. E. and Lin, W. (2017). Conjugate-computation variational inference: Converting variational inference in non-conjugate models to inferences in conjugate models. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Kingma, D. and Welling, M. (2014). Auto-encoding Variational Bayes. *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Lin, W., Khan, M. E., and Schmidt, M. (2019). Fast and simple natural-gradient variational inference with mixture of exponential-family approximations. *Proceedings of the 36th International Conference on Machine Learning (ICML)*.

Martens, J. (2014). New insights and perspectives on the natural gradient method. *arXiv preprint arXiv:1412.1193*.

McGrory, C. and Titterington, D. (2007). Variational approximations in Bayesian model selection for finite mixture distributions. *Computational Statistics & Data Analysis*, 51(11):5352 – 5367. Advances in Mixture Models.

Nguyen, T.-N., Tran, M.-N., and Kohn, R. (2020). Recurrent conditional heteroskedasticity. *arXiv:2010.13061*.

Nott, D. J., Tan, S., Villani, M., and Kohn, R. (2012). Regression density estimation with variational methods and stochastic approximation. *Journal of Computational and Graphical Statistics*, 21:797–820.

Ong, V. M.-H., Nott, D. J., and Smith, M. S. (2018). Gaussian variational approximation with a factor covariance structure. *Journal of Computational and Graphical Statistics*, 27(3):465–478.

Ormerod, J. T. and Wand, M. P. (2010). Explaining variational approximations. *American Statistician*, 64:140–153.

Paisley, J., Blei, D., and Jordan, M. (2012). Variational Bayesian inference with stochastic search. In *International Conference on Machine Learning*, Edinburgh, Scotland, UK.

Ranganath, R., Gerrish, S., and Blei, D. M. (2014). Black box variational inference. In *International Conference on Artificial Intelligence and Statistics*, volume 33, Reykjavik, Iceland.

Rao, C. R. (1945). Information and accuracy attainable in the estimation of statistical parameters. *Bull. Calcutta. Math. Soc*, 37:81–91.

Sato, M. (2001). Online model selection based on the variational Bayes. *Neural Computation*, 13(7):1649–1681.

Tan, L. and Nott, D. (2018). Gaussian variational approximation with sparse precision matrices. *Stat Comput*, (28):259–275.

Titsias, M. and Lázaro-Gredilla, M. (2014). Doubly stochastic Variational Bayes for non-conjugate inference. *Proceedings of the 29th International Conference on Machine Learning (ICML)*.

Titterington, D. M. (2004). Bayesian methods for neural networks and related models. *Statist. Sci.*, 19(1):128–139.

Tran, M., Nott, D., and Kohn, R. (2017). Variational Bayes with intractable likelihood. *Journal of Computational and Graphical Statistics*, 26(4):873–882.

Tran, M.-N., Giordani, P., Mun, X., Kohn, R., and Pitt, M. K. (2014). Copula-type estimators for flexible multivariate density modeling using mixtures. *Journal of Computational and Graphical Statistics*, 23(4):1163–1178.

Tran, M.-N., Nguyen, D. H., and Nguyen, D. (2020a). Variational Bayes on manifolds. Technical report. https://arxiv.org/abs/1908.03097.

Tran, M.-N., Nguyen, N., Nott, D., and Kohn, R. (2020b). Bayesian deep net GLM and GLMM. *Journal of Computational and Graphical Statistics*.

Wand, M. P., Ormerod, J. T., Padoan, S. A., and Frühwirth, R. (2011). Mean field variational bayes for elaborate distributions. *Bayesian Anal.*, 6(4):847–900.

Waterhouse, S., MacKay, D., and Robinson, T. (1996). Bayesian methods for mixtures of experts. In Touretzky, M. C. M. D. S. and Hasselmo, M. E., editors, *Advances in Neural Information Processing Systems*, pages 351–357. MIT Press.

Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

Zhang, F. and Gao, C. (2020). Convergence rates of variational posterior distributions. *Annals of Statistics (to appear)*, 48(4):2180–2207.