Utrecht University
Department of Information and Computing Science

Master thesis

Variational Inference in Stan

First examiner:
Matthijs Vákár

Second examiner:
Gabriele Keller

Student:
Gertjan Brouwer

Daily Supervisor:
Tom Smeding

July 27, 2024

Abstract

This thesis presents a novel, robust, and user-friendly variational inference (VI) algorithm implemented in Stan. As is customary, the algorithm uses the classical mean-field Gaussian variational family and the reparameterization trick to derive a gradient estimator. The novelty lies in the precise optimization routine and convergence criterion. The optimization operates in two phases: the warm-up, where quick progress is made and an appropriate step size is found. This is followed by the main optimization phase, which gradually decreases said step size when reaching stationarity. The process converges when the 2-Wasserstein distance between two Gaussian approximations from two epochs in the optimization phase is below a threshold. This threshold, set by the end-user, offers an intuitive accuracy parameter for convergence, in contrast with other VI implementations.

Experimental results demonstrate the algorithm's robustness and effectiveness across various models from PosteriorDB, with accuracy and runtime largely insensitive to hyperparameter variations. The inclusion of model priors in the initialization phase significantly speeds up optimization, although implementation of this in Stan is still infeasible. The algorithm's reliability is further enhanced by k-hat, although it proved ineffective for detecting untrustworthy posterior approximations.

Contents

1 Introduction

Bayesian statistics have seen an influx of usage in recent years. This can primarily attributed to the availability of more computation power. Specifically, Bayesian inference has seen rapid growth as it has become a crucial part of machine learning. The central objective of Bayesian inference is to determine the posterior probability distribution. Bayes' rule lies at the heart of Bayesian statistics and thus inference. Bayes' rule provides a means to calculate the posterior. However, calculating the posterior is rarely analytically possible and often intractable. Therefore, other methods are required to approximate the posterior probability distribution. Various algorithms have been devised to calculate this approximation. Most notably, Markov Chain Monte Carlo (MCMC) methods have seen great success. More recently, Variational Inference (VI) has been used more.

Markov Chain Monte Carlo (MCMC) methods have been successfully used to solve approximations of the posterior. MCMC algorithms rely on a sampling approach to approximate the posterior distribution by generating samples from a Markov chain that converges to the target distribution over time (asymptotic guarantee). In contrast, Variational Inference (VI) algorithms approximate the posterior by optimizing a simpler, more tractable distribution that is easier to sample from. MCMC and VI can be seen as complementary; whereas MCMC lacks speed on large datasets, VI is much faster. And where MCMC is asymptotically correct, VI's approximations may lack the same level of accuracy.

VI is an idea that tackles the problem of inference through optimization. It uses a simpler distribution to approximate the posterior in question. It minimizes a divergence measure between the variational distribution and the joint probability distribution, usually the Kullback-Leibler divergence. Minimization between distributions is achieved by iteratively adjusting the parameters of the variational distribution such that the divergence measure is minimized. This is typically done with gradient descent, where gradients of the divergence are estimated.

Stan[1] is an open-source state-of-the-art probabilistic programming language for Bayesian statistical modeling and inference. Stan's back-end is written in C++ and is high-performant; it tries to be a general statistical inference engine. This generality can sometimes cause slow convergence or no convergence at all [23]. Stan allows a user to define a Bayesian statistical model in a convenient language. Stan automatically approximates the posterior density with the data using either MCMC, the NUTS sampler by default, or Automatic Differentiation Variational Inference (ADVI) Stan's VI variant.

Automatic Differentiation Variational Inference is still denoted as being experimental in Stan. Stan's implementation of VI is unreliable in convergence and, when converging, might result in an untrustworthy posterior density [3]. Little work has been done to analyze these problems. Implementing diagnostics for posterior densities approximated with ADVI in Stan has yet to be done. Neither has benchmarking Stan's ADVI implementation against a wide variety of models, for example, with PosteriorDB, a benchmarking tool for inference algorithms. All these problems have hindered the adoption of ADVI by Stan users.

Many of ADVI's problems can be attributed to a gap between what is possible in theory and what can be implemented in practice. Recent research on VI seeks to alleviate and solve some of these

[1] Named after Stanislaw Ulam, a co-inventor of the Monte Carlo method

issues [3]. Also, posterior diagnostics such as Pareto-smoothed Importance Sampling (PSIS) and Simulation-Based Calibration (SBC) are now actively researched.

This thesis bridges the gap between said theory and practice by introducing a new VI implementation for VI, which is thoroughly tested with PosteriorDB and benchmarked against Stan's MCMC. Our implementation uses a variety of new concepts from recent literature, such as $\hat{R}$ for VI and PSIS as a diagnostic. Furthermore, we introduce an initialization process by utilizing priors, present a novel warm-up routine that works in general, and introduce a new convergence criterion that requires just one user-defined parameter.

We demonstrate that our implementation is competitive with NUTS in parameter approximation and is exclusively inaccurate in predictable instances, e.g., where the true posterior is multimodal. Additionally, we show that the step size our warm-up phase finds performs better than setting a fixed step size in the optimization phase. Furthermore, preliminary experiments indicate that initialization through prior sampling speeds up the optimization process. Finally, our implementation is more reliable than ADVI as it successfully converges for all PosteriorDB models except three, of which two[2] are neural nets.

This thesis starts with a background chapter on Bayesian inference, MCMC, and VI. It also provides the reader with the recent advancements in VI literature. We continue by explaining Stan and dive into ADVI and why it is still considered experimental. Then, we describe our VI implementation, specifically the initialization, warm-up, and optimization phase. That chapter also mentions diagnostics and provides the algorithm in pseudo-code. The next chapter shows our benchmark against MCMC, a runtime analysis, the efficacy of k-hat, and a robustness test. Finally, we provide concluding remarks on these experiments and describe avenues for future research.

This thesis addresses the core research question driving the project:

- Can we achieve a robust and user-friendly variational inference algorithm in Stan?

From this question, the subquestions follow naturally:

- What causes the current VI implementation in Stan to not converge on certain models?
- What is the impact of the techniques from 2.3.1 on performance compared to MCMC?
- On which models does VI fail even when incorporating said techniques?
- Is the k-hat diagnostic able to recognize failure on these models?
- What is a robust convergence criterion for VI?
- What is the impact of initialization on VIs speed and accuracy?

The Ethics and Privacy Quick Scan of the Utrecht University Research Institute of Information and Computing Sciences was conducted (see appendix B). It classified this research as low-risk with no fuller ethics review or privacy assessment required.

[2]The third due to our initialization being outside the model's optimization space.

2 Background

2.1 Bayesian inference

In statistics, we are often interested in learning a model based on data. In the classical, or frequentist, setting we have some data $\mathcal{D}$ and a model with parameters θ. The likelihood, $p(\mathcal{D} \mid \theta)$, of the data given the parameters is then maximized, commonly known as Maximum Likelihood Estimation (MLE) [4].

In Bayesian statistics, the probability of the parameters given the data is computed instead. This probability, or density, is named the posterior. Calculating the posterior requires specification of the joint probability distribution, $p(\theta,\, D)$, and the data. From a joint density we can compute the posterior of interest with Bayes' rule:

$$\overbrace{p(\theta \mid D)}^{Posterior} = \frac{\overbrace{p(D \mid \theta)}^{Likelihood}}{\underbrace{p(D)}_{Evidence}} \overbrace{p(\theta)}^{Prior}$$

Where the evidence or marginal likelihood, $p(\mathcal{D})$, is an integral of the joint density, or a sum in the discrete case.

$$p(\theta \mid D) = \frac{p(D \mid \theta)\, p(\theta)}{\int p(\theta,\, D)\, d\theta} = \frac{p(\theta,\, D)}{\int p(\theta,\, D)\, d\theta}$$

Unless the posterior density has a closed-form solution, such as in the case of conjugate priors, it is intractable. Thus, we require other methods to compute the posterior density.

2.2 Markov Chain Monte Carlo

One common method for approximating a posterior probability distribution in Bayesian inference is Markov Chain Monte Carlo (MCMC). Stan uses two, namely Hamiltonian Monte Carlo (HMC) and the No-U-Turn Sampler (NUTS). Both algorithms enjoy asymptotic guarantees, meaning that with enough time the algorithm will converge to the true posterior distribution. In practice, however, it is not known for how long one should run the algorithm for it to converge, or it might fail to converge in the case of multimodality, for example. Even if the algorithm converges, the result might still be poor due to, e.g., model misspecification. Regardless of the apparent issues with MCMC it is still very useful and used a lot. We will use MCMC as the gold standard for evaluating VI.

2.3 Variational Inference

Variational Inference (VI) is an alternative strategy to MCMC methods that can result in "good" posterior approximations faster. VI posits an optimization problem by approximating the posterior $p(\theta \mid \mathcal{D})$ with an easy-to-sample density $q_\lambda(\theta)$. The goal is to find the value λ such that $q_\lambda(\theta)$ is closest to the posterior, where closeness is calculated with a **divergence** measure.

The **Kullback-Leibler divergence** is usually picked as a measure of closeness [24, 49].

$$KL(q_\lambda(\theta) \,||\, p(\theta \mid \mathcal{D})) \coloneqq \mathbb{E}_{\theta \sim q_\lambda(\theta)}[\log(q_\lambda(\theta)) - \log(p(\theta \mid \mathcal{D}))]$$

While other divergence measures such as α-divergence exist they are rarely used in practice [49]. We are trying to minimize the KL divergence to find the **variational approximation** $q_{\lambda^*}(\theta)$ in the **variational family** $\mathcal{Q} \coloneqq \{q_\lambda(\theta) \mid \lambda \in \Lambda \subseteq \mathbb{R}^n\}$ such that:

$$\lambda^* \coloneqq \arg\min_{q_\lambda(\theta) \in \mathcal{Q}} KL\left(q_\lambda(\theta) \,||\, p\left(\theta \mid \mathcal{D}\right)\right)$$

The KL divergence can be thought of as having two distinct parts. The role of the expectation $\mathbb{E}_{\theta \sim q_\lambda(\theta)}[\log(p\left(\theta \mid \mathcal{D}\right))]$ is promoting the placement of mass where the posterior density is large. The second part $\mathbb{E}_{\theta \sim q_\lambda(\theta)}[\log(q_\lambda(\theta))]$ is the entropy of the variational family which stimulates spreading of the density. Without entropy, minimizing the KL divergence is equivalent to putting all density at the maximum of the posterior density. Note that the KL divergence is not symmetric. While VI employs $KL\left(q \,||\, p\right)$, the opposite $KL\left(p \,||\, q\right)$ is called expectation propagation [49]. This is a different area of research and not further explored in this thesis.

Even after changing to this optimization problem we still have to deal with the intractability of the marginal likelihood $p(\mathcal{D})$. We can expand the KL divergence to reveal its dependence thereof:

$$\begin{aligned}
&\mathbb{E}_{q_\lambda(\theta)}[\log(q_\lambda(\theta)) - \log(p\left(\theta \mid \mathcal{D}\right))] \\
&= \mathbb{E}_{q_\lambda(\theta)}[\log(q_\lambda(\theta))] - \mathbb{E}_{q_\lambda(\theta)}[\log(p\left(\theta, \mathcal{D}\right))] + \log(p(\mathcal{D})) \\
&= \mathbb{E}_{q_\lambda(\theta)}[\log(q_\lambda(\theta))] - \mathbb{E}_{q_\lambda(\theta)}[\log(p\left(\mathcal{D} \mid \theta\right) p(\theta))] + \log(p(\mathcal{D})) \\
&= -\underbrace{(\mathbb{E}_{q_\lambda(\theta)}[\log(p\left(\mathcal{D} \mid \theta\right) p(\theta))] - \mathbb{E}_{q_\lambda(\theta)}[\log(q_\lambda(\theta))])}_{ELBO} + \log(p(\mathcal{D}))
\end{aligned}$$

Omitting the evidence leaves us with a negative ELBO (**Evidence Lower BOund**), a term coined first in physics literature [49]. Maximizing the ELBO minimizes the KL divergence since the evidence does not depend on λ. The name, evidence lower bound, comes from the fact that it lower-bounds the log marginal likelihood. From the previous equation, we have that:

$$\log(p(\mathcal{D})) = KL\left(q_\lambda(\theta) \,||\, p\left(\theta \mid \mathcal{D}\right)\right) + \text{ELBO}(q_\lambda(\theta))$$

Finding the closest variational approximation to the posterior now only requires the posterior up to a normalizing constant, i.e., the joint density. The joint density is known in our setting[3] thus we are left with finding the variational approximation that maximizes the ELBO. In the early days, Coordinate Ascent VI (CAVI) was used to find the variational approximation [19, 4]. CAVI introduced the **Mean-Field assumption** to statistics from statistical mechanics [31]. The idea of this assumption is to presume mutual independence between model parameters allowing the variational distribution to factorize:

$$q_\lambda(\theta) = \prod_i^D q_{\lambda_i}(\theta_i)$$

The factorized form of the variational distribution allows for iterative updates of the ELBO and forms the basis of CAVI and variational message passing [46]. The choice of a variational family for the factorized parameters is important. The most popular is the use of univariate Gaussians [5, 49, 31]. Picking both the model and the variational family from the same category, e.g. the exponential

[3] We are looking at VI in Stan, see chapter 3, where a joint density is always available.

family, allows for a closed-form solution such as with conjugate priors [19]. However, in most cases, models do not work with conjugate priors and require other methods to solve them.

More recently, so-called black-box methods have been introduced [35, 23]. These black-box methods make no assumptions about the model and seek to maximize the ELBO through (**stochastic**) **gradient** methods.

$$\nabla_\lambda \, \mathbb{E}_{\theta \sim q_\lambda(\theta)}[\log(p\,(\mathcal{D},\, \theta)) - \log(q_\lambda(\theta))\,] \tag{1}$$

Due to the expectation in the gradient, the objective function cannot be computed in general. These black-box methods solve this challenge by rewriting equation 1 as an expectation of the gradient.

The approach we use is called the **reparameterization trick**, the trick is to input the randomness as a parameter instead of the randomness being part of the model [29]. Let us consider equation 1 where $\theta \sim q_\lambda(\theta)$ is a univariate Gaussian: $\mathcal{N}(\lambda_\mu, \lambda_{\sigma^2})$. A valid reparameterization in this case is $f_\lambda(\epsilon) \coloneqq \mu + \sigma \cdot \epsilon$ where $\epsilon \sim \mathcal{N}(0, 1)$.

$$\begin{aligned} &\nabla_\lambda \, \mathbb{E}_{\theta \sim q_\lambda(\theta)}[\log(p\,(\mathcal{D},\, \theta)) - \log(q_\lambda(\theta))\,] \\ = &\nabla_\lambda \, \mathbb{E}_{\epsilon \sim \mathcal{N}(0,1)}[\log(p\,(\mathcal{D},\, f(\epsilon))) - \log(q_\lambda(f(\epsilon)))\,] \end{aligned}$$

This reparameterization is useful because we have removed the dependency on $q_\lambda(\theta)$ from the expectation, allowing us to swap the gradient and the expectation:

$$\mathbb{E}_{\epsilon \sim \mathcal{N}(0,1)}[\,\nabla_\lambda \,(\log(p\,(\mathcal{D},\, f(\epsilon))) - \log(q_\lambda(f(\epsilon))))\,] \tag{2}$$

The resulting equation 2 is the **reparameterization gradient estimator**, which we use to estimate the gradient of the ELBO. Different gradient estimators for variational inference exist [49, 5].

Since this thesis works with Stan we use its automatic differentiation capabilities to compute the gradients inside the expectation of equation 2. The only thing left is calculating the expectation itself, which can be done through Monte Carlo integration [27, 35, 23]. Monte Carlo integration with equation 2 is done by drawing from the standard Gaussian and evaluating the mean of the gradients inside the expectation. This yields a noisy but unbiased estimator of the actual gradient, meaning the expectation of the gradient equals the ELBO [35].

$$\hat{g} = \frac{1}{M} \sum_{m=1}^{M} \text{ELBO}_m \tag{3}$$

Where ELBO_m is the ELBO w.r.t. the m'th standard Gaussian draw, this is where the stochasticity in stochastic VI comes from [49].

With the gradient from equation 3 in hand, we can apply **Stochastic Gradient Descent (SGD)**. SGD iteratively updates the distributional parameters with the computed gradient:

$$\theta_{t+1} \coloneqq \theta_t - \delta_t \cdot \hat{g} \tag{4}$$

Where we denote δ_t to be the **step size** with which $\hat{g}$ is multiplied. In general, we can not always find a global optimum and instead seek to find a **stationary point**. Meaning that the gradient of the ELBO is zero or sufficiently close to zero. The convergence to a stationary point can be

guaranteed, aside from some criteria on the distribution and variational family, when the step size sequence, $\{\delta_t\}$, adheres to the Robbins-Monro criterion: [37] It states that the sum of the step size sequence must be infinite to ensure that at each step it continues to learn from new data. Secondly, the sum of the squared step sizes must be finite. This ensures the gradual decrease in step size, allowing the distributional parameters to settle at a stationary point:

$$\sum_{t=1}^{\infty} \delta_t = \infty \qquad\qquad \sum_{t=1}^{\infty} \delta_t^2 < \infty$$

In practice, more elaborate SGD routines are used, e.g., not only using the first but also the second-order derivative to guide the optimization [50, 49]. Lots of work has also been done on different step size schemes, for example, using an adaptive step size to speed up convergence [23].

2.3.1 Recent advances in Variational Inference

While recent years have seen improvements in *fully automatic* variational inference, widespread adoption has still not occurred. One probable cause is the robustness of black-box VI. This is also reflected in the "experimental" status VI is given in Stan. New research has been trying to achieve fully automatic VI by integrating different VI techniques [3]. These techniques try to answer questions surrounding parameter initialization, step size schedules, gradient estimators, variational families, and importance sampling. Their experiments show promising improvements when combining techniques.

Adaptive step size Picking a step size to run an optimization process with, is a notoriously hard problem to solve. The reason is that different problems (usually) require different step sizes, and the optimal step size can even vary within the same problem space [16, 4]. In variational inference literature, we find solutions with fixed step sizes, step sizes picked by some decision rule, or a transient step size based on properties of the optimization process.

The VI implementation in Stan, ADVI, uses a decision rule to pick the starting step size [23]. ADVI runs a short optimization routine with a set of step sizes. The step size with the highest ELBO is then used for optimization. (Note: Most VI optimization processes use an SGD optimizer such as Adam to calculate the descent direction based on the gradient. Such optimizers tend to scale with their gradient and their history, which can, in part, be seen as an adaptive step size. However, all these optimizers require an initial step size to be chosen.). Stan's ADVI implementation then uses regular SGD with an adaptive scheme to change the step size during optimization. This scheme considers the initial step size and keeps a decaying memory of past step sizes from previous optimization steps. It is similar to Adam, but the authors introduced this as an alternative [23, 20].

Agrawal et al. use a similar technique where they pick a step size $\delta \in \frac{0.1}{d}\{1, B^{-1}, B^{-2}, B^{-3}, B^{-4}\}$ where d is the dimension of the parameter space and B is a decay constant, e.g. $B = 4$. For each δ, optimization with Adam is run for a fixed number of iterations, 30,000 in the experiment. Agrawal et al. opted to use the step size with the best average ELBO for the 30,000 iterations instead of comparing against a test set based on the parameters obtained by running the VI for each step size. Empirically, this gave more robust results. Dhaka et al. use a fixed step size, 0.3, but half it each time the optimization process satisfies some conditions [11].

Zhang et al. first introduced a combination of both techniques, splitting optimization into transient and optimization phases. The transient phase selects the starting step size, and the optimization phase reduces this when certain conditions are met to satisfy the Robins-Monro criteria. They fit a linear regression on the training loss in the transient phase. If the slope of this regression goes below a threshold, it switches to the optimization phase with the current step size. When the slope is above the threshold, the step size is increased by an order of magnitude, and the process continues. Their optimization process uses a variant of the Armijo rule, which is shown to perform poorly for variational inference [25]. Furthermore, their experiments are all done on neural networks and might not carry over to variational inference [51].

Optimizers Using plain Stochastic Gradient Descent as an optimizer has been replaced by adaptive optimizers such as Adam, Adagrad, and Adagrad's successor RMSProp. Adagrad is the first widely adopted optimizer that uses a different learning rate[4] per dimension. Due to Adagrad's long-term issue with a diminishing learning rate, RMSProp was introduced to fix this stagnation. Later, Adam used the momentum technique from RMSProp and adjusted the step direction based on the first and second moment of the gradient [15, 20].

No seminal work has been published on stochastic optimizers for variational inference, also confirmed by Dhaka et al. [3]. Published comparison studies have tested various optimizers with different algorithms and models [48, 30]. Yazan et al. studied many related experimental works on optimizers and concluded that diverse findings suggest no single optimizer is best in general. Further, they mention that using any adaptive strategy, e.g., Adam, works best over plain SGD. Finally, they conclude by saying the effect of the initial learning rate and initialization of the models have a greater effect than the optimizer [48].

In the VI space, we also observe the diversity of optimizers used.

- *BBVI*: Adagrad [35]
- *RAABBVI & RABBVI*: RMSProp, avgRMSProp, avgAdam, Natural Gradient Descent (NGD), stochastic quasi-Newton and Windowed Adagrad (default in PyMC3) [11, 45, 19]
- *ADVI*: RMSProp is the foundation for their optimizer, which they combine with a set of other techniques [23].

This diversity once more suggests no single optimizer is clearly best.

Entropy When using the reparameterization trick on the KL-divergence, we get equation 2. The last term in the expectation of this equation is $log(q)$. Since the expectation is with respect to q, it is the expectation with respect to q of $log(q)$, in other words, the entropy of q. Since we use mean-field Gaussians as a variational family, this entropy term can be solved analytically, i.e., zero and one for the mean and standard deviation gradient, respectively. However, Agrawal et al. suggest estimating entropy using the reparameterization trick can be beneficial. Using the reparameterization trick on the entropy term results in the Stick the Landing (STL) estimator:

$$\nabla_\lambda \mathbb{H}(q_\lambda) = \mathbb{E}_{\epsilon\sim\mathcal{N}(0,1)}[\underbrace{\nabla_\lambda \log(q_\theta(f(\epsilon)))_{\theta=\lambda}}_{\text{STL estimator}}] \tag{5}$$

[4]Learning rate and step size are sometimes used interchangeably, we refer to the step size that an optimizer adapts as the learning rate and the step size that is input as a parameter as the step size.

Where $\mathbb{H}$ is the entropy part of equation 2 [3]. Agrawal et al. show empirically that the STL estimator performs better than the closed-form entropy [38, 3]. It is, however, unclear why this would show better performance. Roeder et al. suggest that it might be due to the entropy depending directly on the variational family without considering the data.

Distributional parameter initialization Initialization can have a large positive and negative impact on performance. ADVI initializes with a $uniform(-2, 2)$ distribution, while most literature on VI uses a standard Gaussian initialization. Agrawal et al. tried using Laplace's method [3]. This method finds a mode of the posterior distribution, which is set as the mean of a Gaussian distribution. An approximation of the standard deviation of the Gaussian is computed from the Hessian of the log-posterior at this mode. An immediate issue with this is that it can initialize at a mode that leads to worse local optima [3]. In their experiment, Laplace initialization helped and harmed an equal fraction of models. Furthermore, they mention no existing studies on initialization surrounding VI, a claim made in 2020. Although no studies exist and their experiment does not show significant results, they argue that a beneficial initialization scheme probably exists and is worth researching.

Stan initializes all model parameters randomly from a $uniform(-2, 2)$ distribution. The paper on Pathfinder, an algorithm that was recently added to Stan, also tested $uniform(-15, 15)$ distributions and found it explored a larger region. This test was only done on a single model and showed ADVI outperformed Pathfinder when using the default parameter initialization [50]. However, no initialization scheme was given that works for any model. Furthermore, both initialization forms will not generally work since models whose parameters lie outside these ranges exist. Therefore, these initialization schemes are not optimal. Hence, the default initialization in mean-field and full-rank variational inference literature still is the standard normal [5, 49].

Importance (Re)sampling Importance sampling (IS) is a method to enhance the accuracy of a sample from a proposal distribution, i.e., q_λ, to give a set of draws that more accurately represent the target distribution, i.e., $p(\theta \mid \mathcal{D})$. This is done by multiplying a draw with its normalized importance weight:

$$\frac{\sum_{s=1}^{S} w_s \cdot \theta_s}{\sum_{s=1}^{S} w_s}, \qquad w_s = \frac{p(\theta_s, \mathcal{D})}{q_\lambda(\theta_s)} \tag{6}$$

With θ_s being S draws from the proposal distribution, i.e., the variational approximation. The rationale behind importance sampling is that we need to adjust for q_λ oversampling and undersampling areas compared to p. Importance sampling in VI has been successfully used by many authors, usually gaining accuracy on a subset of tested models and never losing accuracy [49, 3, 13]. However, improvement is not guaranteed since a mismatch between the true posterior and the approximation can occur, causing certain regions to be over-/underrepresented.

Importance resampling (IR) is a subsequent step after importance sampling. Importance resampling resamples from the set of samples using their normalized weights as a probability distribution. The advantage of IR over IS is that the samples can be immediately used instead of the original and treated as samples from the posterior distribution [21].

There are two primary use cases for IS and IR in variational inference: plain sampling, which benefits the end user by providing better samples, and changing the ELBO for one that uses IS, resulting in a more robust variational approximation [3, 47, 44].

The *Importance Weighted ELBO (IW-ELBO)* introduced by Burda et al. is one such IS ELBO that serves as a replacement for the classic ELBO [6, 14, 3].

$$\mathbb{E}_{q_\lambda(\theta_1)\cdots q_\lambda(\theta_m)}[\log\left(\frac{1}{M}\sum_{m=1}^{M}\frac{p\left(\mathcal{D},\,\theta_m\right)}{q_\lambda(\theta_m)}\right)] \tag{7}$$

Where $\theta_1 \cdots \theta_m$ are iid samples from the variational distribution. When $M = 1$, the IW-ELBO reduces to the normal ELBO while increasing M will result in approaching $log(p(\mathcal{D}))$ asymptotically. The IW-ELBO can be used in two ways during the training phase. Firstly, instead of using a single draw we sample M draws from the variational approximation. Then, with importance resampling, we draw a single sample and use this as if we are doing regular ELBO optimization [14, 13]. As described under *Divergence measures*, the KL-divergence is sensitive to outliers. Using some form of IS or IR, like the importance resampling done here with the IW-ELBO, will reduce this sensitivity [3, 13].

Secondly, we can use the IW-ELBO in the training process instead of the ELBO, but this requires a gradient estimator such as equation 2. Tucker et al. introduced the *Doubly Reparameterized Gradient Estimator (DReG)* which handles cases where M is large well, an issue the Rainforth et al. estimator struggled with, and is now considered the defacto standard gradient estimator for the IW-ELBO [41].

$$\begin{aligned}
&\nabla_\lambda \mathbb{E}_{q_\lambda(\theta_1)\cdots q_\lambda(\theta_m)}\left[\log\left(\frac{1}{M}\sum_{m=1}^{M}\frac{p\left(\mathcal{D},\,\theta_m\right)}{q_\lambda(\theta_m)}\right)\right] \\
&= \mathbb{E}_{q_{(\mathbf{0},\mathbf{I})}(\eta_1)\cdots q_{(\mathbf{0},\mathbf{I})}(\eta_m)}\left[\sum_{m=1}^{M}(\hat{w}_m)^2\,\nabla_\lambda \log\left(\frac{p\left(\mathcal{S}^{-1}(\eta_m),\,\mathcal{D}\right)}{q_\theta(\mathcal{S}^{-1}(\eta_m))}\right)\right]\Bigg|_{\theta=\lambda}
\end{aligned} \tag{8}$$

Where $\hat{w}_m = \frac{w_m}{\sum_{i=1}^{M} w_l}$ and $w_m = \frac{p\left(\mathcal{D},\mathcal{S}^{-1}(\eta_m)\right)}{q_\lambda(\mathcal{S}^{-1}(\eta_m))}$. The central idea here is that the variational parameters are held constant to reduce the variance. Although Domke et al. found that DReG improved nearly 30% of their tested models, using plain IW-sampling combined with the STL-estimator gave better performance [3].

Sampling can benefit from using IS and IR. However, problems arise when the importance weights have infinite variance. Solutions, such as the truncation rule, guarantee finite variance under mild conditions but can have a large sample bias [44, 12]. Vehtari et al. suggest fitting a three-parameter generalized Pareto distribution (GPD) to the importance weights [47, 44]. The CDF of this GPD is then used to replace the largest M importance weights with a set of well-spaced importance weights consistent with the GPD's tail. Doing this ensures the new samples, Pareto Smoothed Importance Samples (PSIS), have finite variance. One of the three parameters of the GPD is the shape parameter $\hat{k}$, which can be used as a diagnostic tool, further discussed in section 2.3.2.

Iterate averaging Each optimization step of black-box VI produces a variational approximation q_{λ_t}. Eventually, under Robins-Monro conditions, these variational approximations will converge to an exact solution q^*_λ. However, after a finite number of iterations, the variational approximation is not close, in general, to the exact solution [11]. A solution to this is to average the variational

approximations over time [33].

$$\overline{q_\lambda} \coloneqq T^{-1} \sum_{i=1}^{T} q_{\lambda_i}, \qquad Var(\overline{q_\lambda}) = \frac{1}{T^2} \sum_{i=1}^{T} Var(q_{\lambda_i})$$

This approach reduces the variance of the approximation because the expected value of q_{λ_t} is equivalent to q^*_λ. Dhaka et al. first utilized this technique for VI [11, 12] and dubbed it iterate averaging. Two implicit choices that arise when doing iterate averaging are when to start the averaging process and how many approximations to average. The first requirement intuitively means that we want the optimization process to have somewhat stabilized before the averaging process. This helps ensure the approximations are closer to the exact solution. Secondly, we want to iterate over a large enough portion of the variational approximations to ensure a low enough variance and, thus, robustness to noise.

Picking good values for when to start averaging and the total number to average is experimentally done by Dhaka et al. and later expanded on by Welandawe et al. [11, 45]. Both start averaging after the optimization reaches stationarity. We can check stationarity using the $\hat{R}$ diagnostic, which we discuss further in the optimization phase section 5.3. Choosing a range of approximations to average over is done by calculating the effective sample size (ESS) and the Monte Carlo standard error (MCSE) of the variational parameters over some range. That range is determined by checking if the ESS is greater and the MCSE is smaller than some threshold. The underlying principle for this to work is that we can see the series of variational approximations, *after* stationarity, as a Markov chain. Allowing us to quantify the accuracy of the iterate average through ESS and MCSE.

Divergence measures Traditionally, VI methods use the KL-divergence measure to gauge the distance between the posterior distribution and the variational approximation. However, classic KL-divergence has some shortcomings:
Not robust to outliers: The logarithm in the KL-divergence magnifies the outliers. Assume the one-dimensional true posterior $p\left(\theta \mid \mathcal{D}\right) \sim \mathcal{N}(\mu_1, \sigma_1^2)$ and the approximating mean-field Gaussian distribution $q_\lambda(\theta) \sim \mathcal{N}(\mu_2, \sigma_2^2)$. Assume that $\sigma_1^2 = \sigma_2^2 = 1$, $\mu_1 = 0$ and $\mu_2 = \epsilon$. Where ϵ illustrates the outlier. Then the KL-divergence between the Gaussian distributions becomes $\mathrm{KL}\big(\mathcal{N}(\mu_2, \sigma_2^2) \,||\, \mathcal{N}(\mu_1, \sigma_1^2)\big) = \frac{\epsilon^2}{2}$. This shows a quadratic dependency on ϵ; therefore, a small change can significantly impact the value of KL [28]. Although the quadratic dependency remains, using importance sampling can reduce the impact of outliers 2.3.1.
Underestimation of uncertainty: Another, more prominent issue with the divergence is its tendency to underestimate the variance of the variational parameters [28, 49, 5]. This is due to the KL penalizing the placement of probability mass on regions of the variational approximation where the posterior has a lower mass, i.e., the ratio $\frac{q_\lambda(\theta)}{p(\theta \mid \mathcal{D})}$ becomes large. However, when swapping p and q in the KL-divergence, this does not occur. This asymmetry is called mode-seeking behavior and causes the estimated variance to be lower than the true variance [4, 19, 5, 49].
Multimodality: Another matter is the case of multimodality and the KL-divergence's inability to capture this correctly. This is partly due to the mode-seeking behavior, which causes the optimization to focus on the largest mode. Usually causing a collapse into a single mode [35, 5]. Secondary is the choice of variational family, $\mathcal{Q}$, in our case, a Gaussian family. Which is unimodal and, therefore, unable to capture a posterior with a multimodal nature. Solving this is usually done by changing out the variational family for one that can handle multiple modes [12]. Or using the exclusive KL-divergence, as opposed to the inclusive KL-divergence we use here, that exhibits mass-

covering behavior. And will thus capture multiple modes with a unimodal variational approximation [28].

These defects have been an area of research in recent years, which resulted in different and new divergence metrics being employed for VI [28, 49]. Which culminated in a generalization of divergence measures that still work in a black-box setting [49]. The KL-divergence is a special case of the so-called α-divergence:

$$D_\alpha(p \mid\mid q) = \frac{1}{\alpha - 1} log \int p(\mathcal{D})^\alpha q(\mathcal{D})^{1-\alpha} d\mathcal{D}$$

Where $\alpha > 0,\ \alpha \neq 1$. The α term is an interaction between the mass-covering effect, $\alpha < 1$, and the zero-forcing effect, $\alpha > 1$. An $\alpha < 1$ tends to penalize assigning low probability in the variational approximation where the posterior has high probability, which is useful when in a multimodal setting. While $\alpha > 1$, it penalizes assigning probability mass where the posterior has low or zero mass. This results in more mass on high-mass areas of the posterior. When $\alpha \to 1$, it becomes the KL-divergence; hence, it is a generalization thereof.

The α-divergence itself is a class of the f-divergence:

$$D_f(p \mid\mid q) = \int q(\mathcal{D}) f\left(\frac{p(\mathcal{D})}{q(\mathcal{D})}\right) d\mathcal{D}$$

Choosing $f(w) = w\ log(w)$ where $w = \frac{p(\theta, \mathcal{D})}{q_\lambda(\theta)}$ results in the KL-divergence. $f(w) = (w-1)^2$ is the χ^2-convergence and $f(w) = \frac{(w^\alpha - w)}{\alpha(\alpha-1)}$ corresponds to the α-divergence.

A review of these generalizations shows marginal improvements [12]. However, experiments are often done in combination with other recent improvements, causing dilution of the results.

Variational family The choice of the variational family can significantly impact the accuracy of the final approximation. The default variational family for fully automatic variational inference is the mean-field Gaussian [49, 35, 3]. Due to the nature of being Gaussian, the mean-field family is not expressive enough to capture any and all imaginable posteriors. Especially when the true posterior is thick-tailed, skewed, or multimodal [36]. Several continuations of the mean-field Gaussian have been tested, such as the full-rank Gaussian family [49]. However, these tend to have a marginal gain only. More promising results come from the corner of normalizing flows, a type of invertible neural network. Early analysis by Domke et al. suggests improvements may be made with normalizing flows, but the results are preliminary [3]. Dhaka et al. show normalizing flows to be effective, but not straight out-of-the-box since they require problem-specific tuning [12].

2.3.2 Diagnostics

Variational approximation using VI generally has two sources of problems. First, sometimes it converges slowly or does not converge at all. Second, it may converge to a poor approximation of the true posterior due to the variational family being too limited or the true posterior having thick tails, being skewed, or being multimodal [44, 47, 42]. Therefore, it is helpful, or even necessary, to communicate back to a user of Stan whether the approximation is faulty. One caveat with Bayesian inference is the uncertainty in the fit of models [10]. Even when the algorithm converges, we can not be certain the results are correct. We can use diagnostics to convey that something did not

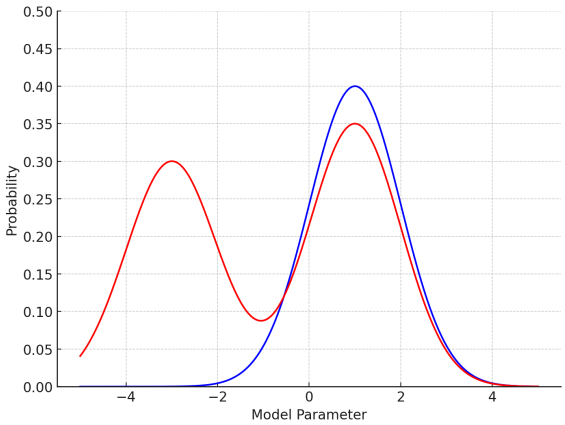

Figure 1: Example Gaussian fit (blue) of a multimodal posterior (red), source: Appendix A

go well, but when these diagnostics all pass, there is still a possibility that the model is incorrect or inference failed. Nonetheless, we still want diagnostics to increase confidence and alert us when inference go awry.

k-hat One such metric is k-hat, $\hat{k}$, which is the shape parameter of a generalized Pareto distribution fit on the importance ratios of a sample from the VI approximation [47, 44]. Vehtari et al. and Yao et al. suggested using k-hat as a diagnostic tool to detect failure in the variational approximation to approximate the true posterior [44, 47]. The PSIS diagnostic works by running VI to obtain a variational approximation, $q_\lambda(\cdot)$, from which S samples are drawn, θ_s. From these samples S importance ratios are calculated: $r_s = \frac{p(\theta_s, \mathcal{D})}{q_\lambda(\theta_s)}$. On these importance ratios, a generalized Pareto Distribution is fit, which itself has three parameters, one of which is the $\hat{k}$ parameter, the parameter used to diagnose problems. Vehtari et al. and Yao et al. both give a guideline for what $\hat{k}$ means in the context of a variational approximation.

- If $\hat{k} < 0.5$, we can conclude that the variational approximation is close to the true posterior.
- If $0.5 < \hat{k} < 0.7$, it indicates that the variational approximation is not ideal but practically useful.
- If $\hat{k} > 0.7$, the variational approximation is flawed and can not be used as an accurate replacement for the posterior.

Providing the k-hat diagnostic after fitting a variational approximation can give a user some indication of the reliability of the variational approximation of their posterior. However, determining the exact problem when $\hat{k} > 0.7$ is still hard and can be due to many causes, such as model misspecification, multimodal posterior, wrong reparameterization, optimized for too few iterations, and too large a learning rate.

Dhaka et al. show that the $\hat{k}$ diagnostic grows rapidly when working with samples from a high-dimensional model [12]. In a figure from their paper, figure 2, the rapid growth of the $\hat{k}$ diagnostic can be seen for the inclusive KL-divergence, also known as just the KL-divergence. They compared it to a form of KL-divergence, exclusive KL-divergence, which tends to put probability mass on

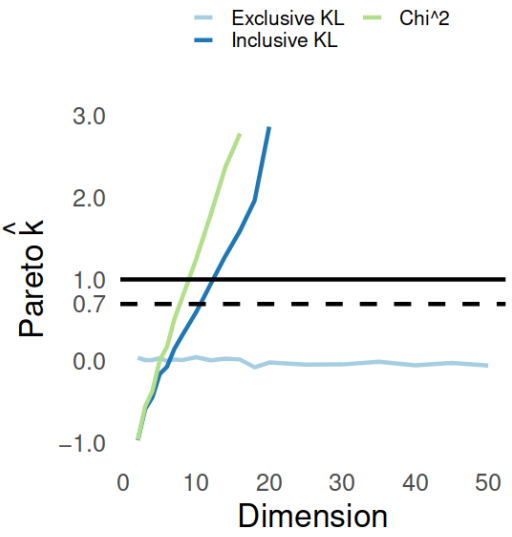

Figure 2: $\hat{k}$ rapid growth - Source: [12]

the dominant mode of the posterior. This results in underestimating the uncertainty but having a more robust $\hat{k}$. However, the authors mention that optimizing with the exclusive KL-divergence is harder in practice [12].

R-hat, $\hat{R}$, is traditionally a diagnostic to measure convergence in MCMC algorithms [18, 16]. For MCMC, $\hat{R}$ provides a measure to determine if a Markov chain has converged. The regular way of calculating $\hat{R}$ is by computing the variance of each chain and dividing it by the between-chain variance. If $\hat{R} < 1.01$, it is considered a converged chain [45].

Dhaka et al. transfer this idea over to variational inference. They store this new approximation at each step, i.e., variational parameter update. They treat this series of approximations as a Markov chain to determine when to stop iterating their variational algorithm. They use five different window sizes to look back in the chain, i.e., variational approximations, and determine the $\hat{R}$ for each window. Specifically, they compute the $\hat{R}$ for the mean of each approximated parameter in the variational distributions. When one of these $\hat{R}$ is below 1.1, they call the variational algorithm converged [11, 45]. Dhaka et al. primarily use $\hat{R}$ to determine stationarity; however, they also mention it can conversely detect optimization problems. Doing this requires multiple chains or, in our case, multiple runs of the variational algorithm. This allows the different instances to end up in distinct optima indicated by a large $\hat{R}$. The authors mentioned this as future work but have not published on this as of yet [45].

3 Stan

Stan is an easy-to-use high-performance probabilistic programming language and Bayesian statistics tool. The main goal is to provide an easy way to describe Bayesian models and provide a suite of inference algorithms for fitting these. Stan exposes several interfaces, many of which are libraries for various programming languages. Through these interfaces models can be fit with a variety of algorithms, such as MCMC algorithms, HMC and NUTS, as well as ADVI, a variational inference algorithm, and a maximum likelihood estimation algorithm. Describing models is done with a high-level modeling language that allows one to imperatively define a (log) joint probability over a set of parameters conditioned on data and possibly constants. This high-level language, sometimes also confusingly called Stan, is transpiled down to C++ code, which is subsequently compiled and linked into an executable for fitting [7].

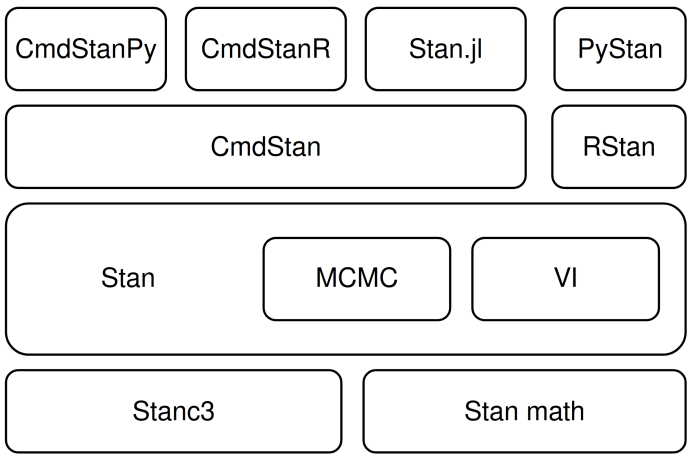

Figure 3: Stan architecture

The Stan Math Libary is reverse mode automatic differentiation library[5]. The library includes functions for matrix operations, linear algebra, solving differential equations, and its extensive support for probability distributions [8]. Differentiation is the main use case of the math library. The HMC and NUTS samplers both use gradients to guide their parameter updates, the BFGS optimizer also uses gradients to search around the space, and VI also makes heavy use of gradients. The library is very usable due to its direct, clean, functional, and simple interface while also being efficient through lots of memory optimizations, caching, and expression templates [8].

The Stanc3 compiler transpiles a *.stan* file into C++ header file for compiling and linking. It is written in OCaml, a functional language, and is the main compiler for Stan. The compiler works through lexing, parsing, type-checking, and optimizing the *.stan* file. The resulting C++ header file contains functionality to calculate the log probability and its gradient[6] using Stan's math library.

[5] It also supports forward and mixed mode, but the reverse mode is mostly used.
[6] More complex models might expose more functionality.

3.1 Stan language

The Stan language provides an easy-to-use way to describe probabilistic models, specifically, how the joint probability is defined. A strength of Bayesian inference is the ease with which hierarchical models can be solved [16]. Here we work through an example of defining, fitting, and working with a hierarchical model in Stan.

Eight schools is a famous study on the effects of specialized coaching programs for the SAT-V (Stochastic Aptitude Test-Verbal)[7] in eight high schools in the United States. This study is often used to convey the power of hierarchical models. The outcome of the test, as shown in table 1, is the efficacy of the coaching programs per school on the SAT-V score. The SAT-V score can vary between 200 and 800 points with a mean of 500 and a standard deviation of around 100 points [16], with coaching having an effect of 28 ± 15 points in school A and -1 ± 9 in school E.

School	Estimated effect (y_j)	Standard error (σ_j)
A	28.39	14.9
B	7.94	10.2
C	-2.75	16.3
D	6.82	11.0
E	-0.64	9.4
F	0.63	11.4
G	18.01	10.4
H	12.16	17.6

Table 1: The difference in SAT-V score for the treatment group with respect to the control group and the corresponding standard error.

Inference with a non-hierarchical model is more common in a non-Bayesian setting [17]. Two ways to approach non-Bayesian inference with the data are using a separate estimate for each school, or pooling all data together and having a single estimate. Doing a simple normal analysis per school yields a 28.5 mean and 14.9 standard error in school A. Doing a pooled estimate gives a mean of 7.7 and a standard deviation of 4.1. In the separate analysis, it implies that the chance of the efficacy being greater than 28.5 is 50% while in the pooled estimate, the efficacy has a 50% chance of being below 7.7. Considering the other seven schools, the efficacy being greater than 28.5 is improbable, while it being less than 7.7 is not likely either [16, 17]. Both estimates leave something to be desired and are a springboard for hierarchical analysis.

With a hierarchical model, each school gets its own mean, which is drawn from a normal distribution with a global mean and variance, which are estimated from the data. When the effects between schools are similar, the global variance will be lower; by contrast, the variance will be higher when the effects are dissimilar between schools. The model in figure 4 encodes this hierarchical model for the eight schools. The global mean, μ, and global variance, τ, are both sampled from a prior uniform distribution [16]. Per school, the mean efficacy follows a normal distribution with parameters μ and τ. The sampling follows a normal distribution with the per school mean and standard deviations that are known from, and equal to, table 1.

Writing this hierarchical model in the Stan language is done in a *.stan* file, see figure 5. Stan models

[7] The SAT is a standardized acceptance test for higher education in the USA.

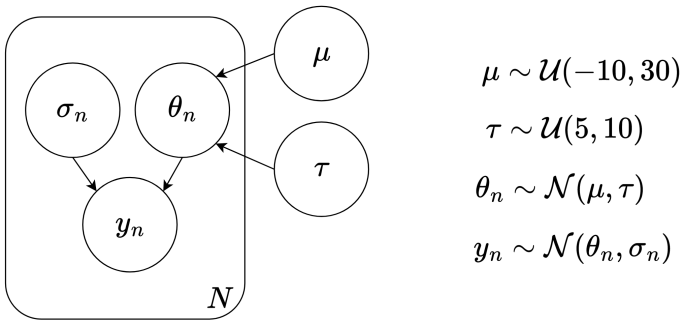

Figure 4: Eight schools hierarchical model

```
data {
    int N;          // number of observations
    int sigma[N];   // standard deviation of each school
    int y[N];       // estimated coaching effect
}

parameters {
    real mu;
    real<lower=0> tau; // standard deviation must be positive
    real theta[N];
}

model {
    mu ~ uniform(-10, 30);
    tau ~ uniform(5, 10);
    theta ~ normal(mu, tau);

    // likelihood
    y ~ normal(theta, sigma);
}
```

Figure 5: Eight schools hierarchical model specified in .stan

are, in essence, a specification of the (log) joint probability distribution, also called joint density. The joint density is usually a combination of priors and likelihoods, e.g., τ being a prior and θ_n being a likelihood based on τ. Stan uses this joint density to fit a posterior often with MCMC methods like NUTS; in the case of VI, it is used to minimize the divergence between itself and the variational family.

A *.stan* file in its most stripped-down form consists of a *data*, *parameters*, and *model* block. Describing what data is input, which parameters to learn, and how the model, i.e., joint density, is specified, respectively. For most use cases, as with the eight schools example, these blocks will suffice. Stan, however, provides the blocks *generated_quantities*, $transformed\_data$, and $transformed\_parameters$ as well. These are useful for generating new data from scratch or existing data before fitting or specifying new variables based on estimated parameters.

```python
import os
import numpy as np
import pandas as pd
from cmdstanpy import CmdStanModel

stan_file = os.path.join(os.getcwd(), 'models/8schools.stan')
model = CmdStanModel(stan_file=stan_file)

y = [28.39, 7.94, -2.75, 6.82, -0.64, 0.63, 18.01, 12.16]
sigma = [14.9, 10.2, 16.3, 11.0, 9.4, 11.4, 10.4, 17.6]

fit = model.sample(data={"N": len(y), "y": y, "sigma": sigma}, show_console=True)
print(fit.summary(percentiles=(5, 25, 50, 75, 95)))
```

Figure 6: CmdStanPy Python script to fit eight schools model

3.2 Stan interface

Working with Stan is done through a package; these exist for various programming languages that directly interface with Stan or from the command line via CmdStan (Pronounced: command stan). The interfaces PyStan and RStan directly communicate via memory with the Stan C++ library, whereas CmdStanPy, CmdStanR, and Stan.jl interface through CmdStan[8].

We fit the eight schools model with the CmdStanPy interface. Figure 6 shows the Python script to run the eight schools stan model from figure 5. The data block from the *.stan* file has a one-to-one correspondence with the data parameters passed through the sample function in Python. The data are defined from and equal to the data in table 1. After fitting, which is by default done with NUTS, the summary of the fit is displayed, see figure 7. The summary yields the fitted parameters μ, τ, and the mean for each school, specified in the parameters block in the *.stan* file and $lp\_\_$, which is the log density up to a constant.

	Mean	MCSE	StdDev	5%	25%	50%	75%	95%	N_Eff	N_Eff/s	R_hat
lp__	-19.94540	0.083483	2.32935	-24.067300	-21.293000	-19.71860	-18.25270	-16.57510	778.527	4325.15	1.00434
mu	8.12056	0.204876	5.07036	-0.040864	4.690840	8.13386	11.50470	16.51510	612.488	3402.71	1.00966
tau	7.22880	0.055536	1.40284	5.209970	6.005890	7.09902	8.36179	9.63528	638.072	3544.84	1.00940
theta[1]	11.75250	0.245197	7.57195	-1.092010	6.637100	11.94010	16.77270	24.28610	953.642	5298.01	1.00428
theta[2]	7.81880	0.234616	6.89798	-3.840830	3.192160	8.00204	12.32920	19.17150	864.428	4802.38	1.00227
theta[3]	6.42365	0.272177	7.87602	-6.482750	1.122960	6.26245	11.64090	19.42450	837.359	4651.99	1.01032
theta[4]	7.56263	0.230298	6.77786	-3.505420	3.140760	7.56485	11.83790	19.03600	866.173	4812.07	1.00434
theta[5]	4.92018	0.203537	6.59031	-5.601330	0.306374	4.89699	9.56225	15.75510	1048.400	5824.42	1.00078
theta[6]	6.13026	0.234216	7.19697	-5.563760	1.182500	6.10554	11.07040	17.66470	944.203	5245.57	1.00488
theta[7]	11.22750	0.238308	6.91910	0.070579	6.402710	11.11200	15.90210	22.92080	842.990	4683.28	1.00668
theta[8]	8.93106	0.255290	8.03047	-4.422330	3.735790	9.03898	14.14910	21.73010	989.494	5497.19	1.00664

Figure 7: Eight schools summary after fitting with MCMC

[8]Numerous wrappers around CmdStan exist for various languages, but these are the main three

4 ADVI in Stan

Automatic Differentiation Variational Inference (ADVI) is the VI implementation in Stan based on the 2015 paper by the same name [23]. Since its introduction in 2015, it has remained an experimental implementation, and no quality guarantees are given on the resulting posteriors. This chapter sheds some light on three issues with the current implementation.

4.1 Convergence metric

A difficult-to-solve problem with optimization problems is choosing when to stop the process [4, 16]. ADVI maximizes the ELBO from equation 1. The ELBO is calculated after every N (default is 100) gradient updates. Furthermore, the relative difference between the current and previous ELBO is stored in a circular buffer[9]. Two metrics are calculated based on this circular buffer, namely the average and median. If either of these metrics goes below a tolerance (default 0.01), the optimization process is halted, and the current variational approximation is returned [2, 23].

Kucukelbir et al. did not provide any motivation for using relative difference in their 2015 and 2017 paper on ADVI [23, 22]. But, using the relative difference can cause problems when the absolute changes are large.

$$diff_{rel} \coloneqq \left| \frac{ELBO_t - ELBO_{t-1}}{ELBO_{t-1}} \right|$$

When plugging in $ELBO_t = 1.000.000$ and $ELBO_{t-1} = 990.500$ the relative difference is approximately 0.009 causing the optimization process to halt. However, such large absolute changes after 100 gradient updates indicate the process may not have converged. But switching the relative difference out for the absolute difference causes issues with scale, e.g., an absolute value of 10 is not large when the ELBO is 1.000.000, but it is when the ELBO is 100.

Consider the following Stan model that finds the mean of a data set: $y \sim \mathcal{N}(\mu, 0.01)$. Providing the data $[10, 10, 1, 1, 0, 0, 0, 0, 0, 0]$ should return a value of 2.2 for μ. However, running this with Stan's ADVI returns means ranging from -122 to 320. See figure 8 for a plot of 10 runs. Clearly indicating that the optimization process is halted before convergence has been reached.

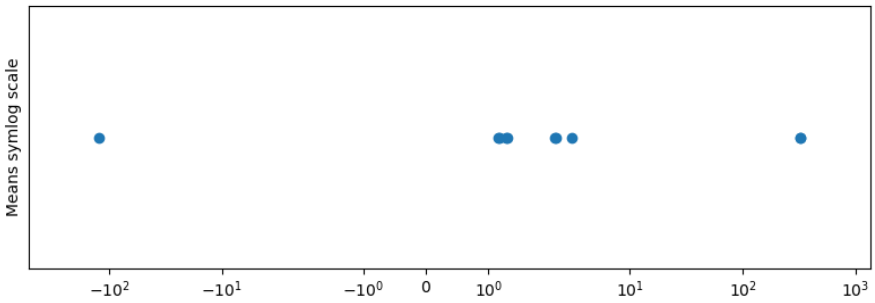

Figure 8: Means of 10 runs of ADVI on the mean model

[9]The circular buffer has a fixed size of 10 (default) that wraps around when the end is reached, thus overwriting the oldest with the newest value.

A secondary problem with using relative difference occurs when the ELBO values are near zero, causing a high relative difference. This can cause the optimization to never halt when the ELBO converges near zero.

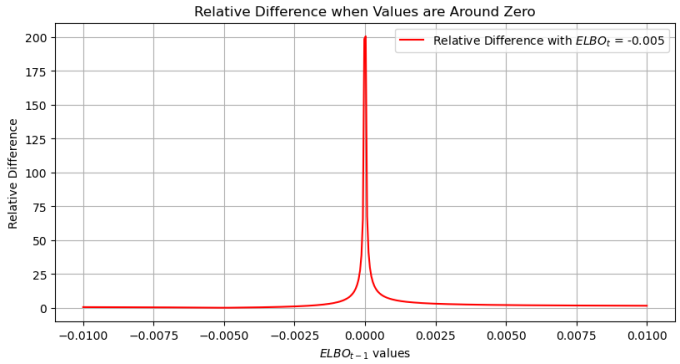

Figure 9: Relative difference with ELBO values near zero

4.2 Convergence subject

The optimization subject is usually the ELBO in variational inference literature, specifically black-box variational inference literature. Naturally, the ELBO is also used to measure convergence, with the relative difference metric in the case of ADVI. However, no convergence criteria on the ELBO will generally halt correctly. This is due to most metrics indicating convergence of the ELBO, while the variational parameters have not stabilized yet [11]. We show this in figure 10, where the first dotted (vertical) line indicates the convergence of the ELBO by the relative difference metric. Therefore, given that convergence checks based on the ELBO are ineffective when using contemporary metrics and considering that the ELBO is merely a means to an end, it is likely fruitful to explore alternative approaches.

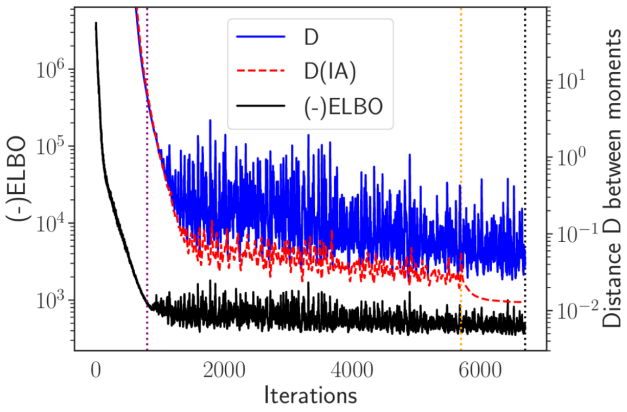

Figure 10: First vertical dotted line indicates convergence of the ELBO - Source: [11].

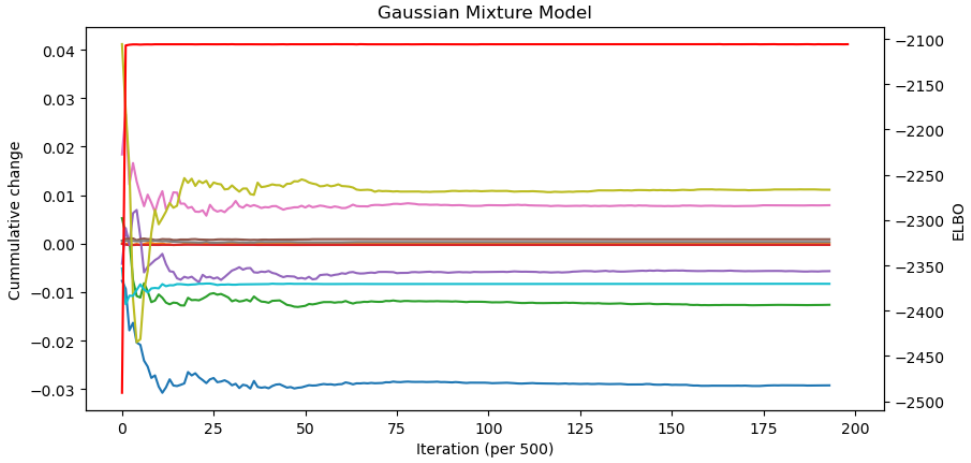

Figure 11: Red line showing the ELBO, others show cumulative parameter change.

Figure 11 shows the ELBO stabilized while the parameters are still changing.

4.3 Step size

Stan's ADVI uses its own optimization scheme, which is a combination of RMSProp and Adam [23]. This scheme uses an adaptive learning rate that takes into account previous gradients, much like Adam, Adagrad, RMSProp, etc [51, 20]. However, much like its constituent components it still requires the specification of an initial step size. Zhang et al. show that using a default step size for a variety of models can result in slow convergence, sometimes orders of magnitude, compared to a well-chosen step size [51].

The existing implementation uses a subroutine to select the initial step size to better fit the scale of the model. The subroutine runs the optimization for N (default 50) iterations for each step size from $[100, 10, 1, 0.1, 0.01]$ and returns the one that yields the largest ELBO. However, once the optimization process starts with the "best" initial step size, it is not adapted afterward. This is because of the adaptiveness of the optimizer, which is the same way RMSProp and the like change the learning rate based on previous gradients. Although the assumption is that the optimizer changes the learning rate sufficiently, the initial step size has a larger impact than presumed [11, 51]. Zhang et al. are the first to tackle this issue but still use the convergence criteria based on the ELBO [51]. While Dhaka et al. and Welandawe et al. also change the convergence criteria to measure the variational parameters [11, 45].

In our analysis, we have identified three primary problem areas so far:

1. Relative difference convergence metric
2. Not using the variational parameters as a convergence subject, but instead the ELBO
3. Fixed step size throughout the optimization process

5 Variational inference

5.1 Variational parameter initialization

Stan's ADVI uses a $uniform(-2, 2)$ distribution to initialize the parameter in the unconstrained space. But most papers on variational inference with the mean-field Gaussians as the variational family use a standard Gaussian as an initialization for the variational parameters [49, 3, 5, 23]. We have also chosen to stick to this, although it is likely not the best initialization strategy.

A problem with initializing with a standard Gaussian is the case of infinite gradients. This issue also persists for Stan's initialization scheme or the $uniform(-15, 15)$ scheme proposed by Zhang et al. [50]. From the PosteriorDB models, at least the models, *sir*, *lotka_volterra*, and *dogs_log*, struggle with this issue. For VI, it is impossible to get the optimization process going because the gradient is negative infinity in Stan. Therefore, we cannot make progress. Changing to a $\mathcal{N}(1, 1)$ distribution for the variational parameters of the *dogs_log* model fixes this. However, for the other two models, it requires the variational parameters of some dimensions to be large, e.g., 10.000 to 100.000, before the gradients are not infinite anymore, *nan* in Stan. Solving this issue requires hand-tuning; we have not found a reliable automatic solution. Agrawal et al. and Domke et al. tried Laplace's method for initialization but an equal fraction of models was helped and hurt. They also mention that thus far no initialization method for VI has been found, a statement made in 2020 [13, 3].

In our opinion, an overlooked idea is to use the priors of the model parameters to initialize the variational parameters. Using priors is a cornerstone of Bayesian statistics, a fact reflected in the models of PosteriorDB as well, where nearly all models are specified with priors [40].

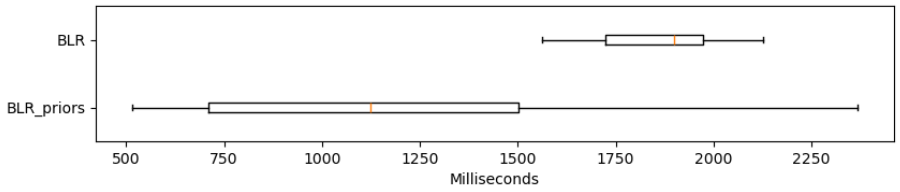

Figure 12: Bayesian linear regression model optimization time with and without initialization with priors. — *blr.stan*

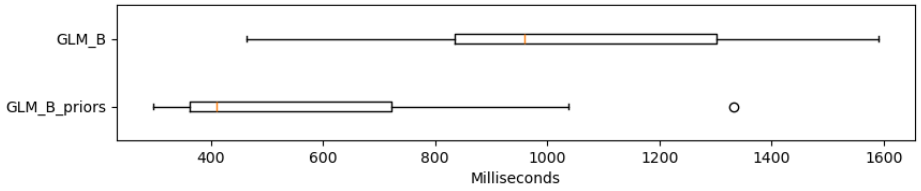

Figure 13: Binomial Generalized Linear Model optimization time with and without initialization with priors. — *GLM_Binomial.stan*

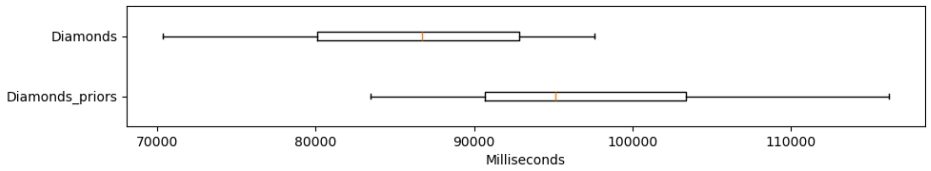

Figure 14: Multiple highly correlated predictors log-log model optimization time with and without initialization with priors. — *Diamonds.stan*

Figures 12 and 13 show a significant speed-up for both the *BLR* and *GLM_Binomial* models when initializing with priors, 38% and 32% respectively, while the *Diamonds* model was hurt slightly by using priors, being 13% slower. Investigation of the true model parameters, as calculated by NUTS, shows that the parameter values with priors are near zero for the *diamonds* model. This means that an initialization with a standard Gaussian is very close to the true parameters, based on sheer luck. While an initialization with the specified priors, $\mathcal{T}(3, 0, 10)$ and $\mathcal{T}(3, 8, 10)$, will deviate at least some from zero in most cases. A model that previously failed to converge due to infinite gradients, *dogs-log*, can now convert due to initialization with priors. This use-case and the speed-up make a good case for using priors as initialization distributions. However, automating this in Stan is not feasible since we have no access to priors, thus requiring huge rewrites of the existing code base. Therefore, all tests have been done by hand for each model, and implementation in Stan remains open; see section 7.

In the preliminaries, we discussed the inclusive KL-divergence (default), which tends to have mode-seeking behavior, and the exclusive KL-divergence, which puts mass on the dominant mode. Since we use the KL-divergence, which is mode-seeking, we have to deal with its sensitivity to initialization since it will not converge to the dominant mode in general [12]. This is both a blessing and a curse since it allows us to detect and work with multimodal posteriors.

Fitting a multimodal posterior with the mean-field Gaussian family is impossible. Therefore, we want to warn the user when we detect a multimodal model posterior since our fit will not reflect this posterior. Generally, it is hard to detect this in a VI setting [49, 5]. But since we know the KL-divergence is sensitive to initialization, we can use this to our advantage. We can choose to run multiple optimization processes (chains) in parallel, each with a different initialization of the variational parameters. Due to KL's sensitivity, these chains might end up in different modes. Calculating the $\hat{R}$ statistic between these chains will indicate this, allowing us to warn the user about multimodality. This has not been implemented and will be addressed in future work (section 7).

Zhang et al., in their paper on Pathfinder, came up with an idea to fit a multimodal posterior with only Gaussian distributions. They do this by running their Pathfinder algorithm in parallel (default 20 parallel instances) until convergence and combining the Gaussian approximation per dimension to form a mixture of Gaussians. This mixture of Gaussians can better approximate posteriors far from Gaussian [50]. We have done no testing but expect this procedure to carry over to variational inference with Gaussian families and postpone this to future work (section 7).

5.2 Warm-up phase

Selecting a proper step size in variational inference, and more generally in optimization problems arising in machine learning, has been a research topic for years [4, 49, 3, 51]. The first paper on black-box VI addressed the issue of step size selection by using an optimizer, such as Adagrad, to have an adaptive learning rate. A tendency perpetuated by the authors of ADVI and RAABBVI as well [35, 23, 11]. However, using only Adam, RMSProp, Adagrad, or other alternatives does not generally work well. Dhaka et al., the authors of RABBVI and RAABBVI, also agree with this assessment. However, they still choose to start with a default step size and lower it based on some scheme, further discussed in section 5.3.

Zhang et al. and Dhaka et al. suggest using a step size scheduler in combination with an adaptive optimizer like Adam. However, only Zhang et al. suggest using a warm-up phase to search for a good starting step size, while Dhaka et al. use 0.3 by default. However, in the same paper, they mention that decreasing the step size too quickly can slow optimization by an order of magnitude, failing to recognize that always starting at 0.3 might cause similar issues, e.g., when the best initial step size is 0.5 or larger. Zhang et al. suggest running the optimization for several iterations before calculating the training loss on which a linear regression is fitted. After each fit, the slope is checked, and if this slope is below a certain threshold, the step size is increased by a factor of 10. If the slope exceeds this threshold, the warm-up is halted, and the current step size is used to start the optimization phase [51]. Although this method was initially intended to optimize deep learning models, the idea is still valid for VI. We have tested the exact same approach on the ELBO but found the step size would increase too fast, causing infinite gradients.

To prevent large step sizes that cause infinite gradients, we use a more dynamic approach. We know from section 4.3 that the ELBO converges well before the variational parameters do. We take advantage of this fact by optimizing until the ELBO converges[10].

We start the warm-up phase with the step sizes: $[10^3, 10^2, 10^1, 10^0, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}]$. For each of these step sizes, we run a fixed number of iterations[11] and calculate the ELBO of the resulting approximation. Then, for the step size with the largest ELBO, we continue this same procedure with that step size, the same step size but doubled, and the same step size but halved. This allows us to grow and shrink the step size according to its performance. When the ELBO stagnates, i.e., the ELBO at the current timestep is within two standard deviations of the previous ELBO, we stop this process.

Now, we start a transient phase where we grow the step size by a factor of two. We begin with the step size from the warm-up phase, run an optimization for 100 iterations, and calculate the ELBO afterward. If the ELBO is within two standard deviations of the stagnated ELBO, we increase the step size by two. Otherwise, we stop the transient phase and start the optimization phase with the current step size divided by two; see algorithm 1.

What this process achieves is quickly reaching a point in the optimization space where the ELBO is more or less stable. This is possible since the ELBO converges before the variational parameters, at least by the most commonly used metrics (see section 4.2). Since the optimization starts with a wide range of possible step sizes, one that fits the model is automatically selected. If the step size had caused an issue, e.g., an infinite gradient, it would have been discarded. Any subsequent step

[10] With more loose bounds than the ADVI convergence criteria.
[11] Refer to the ablation study in section 6 for the exact number of iterations.

size, exactly an order of magnitude smaller or larger, is only chosen if it increases the ELBO more than the current step size. This yields an appropriate step size for the model we are fitting. We can use this step size as the starting step size for the optimization phase. However, we presume we can finish the optimization phase quicker by first increasing the step size. We think this works since if the ELBO converges with step size 0.1 and it converges before the variational parameters, then we are updating the variational parameters with step size 0.1. Thus, if we keep increasing the step size further, by doubling the step size, and the ELBO remains stable, we are doubling the speed with which we update the variational parameters. If this is the case, the best thing to do is to increase the step size until the ELBO becomes volatile again. We presume this works since a stable ELBO means we have a step size that either keeps the variational parameters near a local optimum or increases them steadily. A volatile ELBO means the step size makes changes in the variational parameters that are too radical. In summary, select the step size that yields the best ELBO from a wide range of possible step sizes. Run parallel optimizations of step sizes one order of magnitude away, keeping the best one, and repeat this process for the step size that produces the greatest ELBO. When the ELBO converges, increase the resulting step size by a factor of two until the ELBO becomes volatile again. Use the step size that yielded the last stable ELBO for optimization.

We experimentally show this phase works using a ubiquitous Bayesian linear regression model. Figure 15 and 16 show that using a fixed step size is slower than using our warm-up approach. The efficiency of the optimization routine with a specific step size depends on the model and data. Therefore, hand-picking step sizes perform worse than using a warm-up routine. Running only the warm-up routine and not the additional transient phase will be slower, as shown in figures 17 and 18. An attentive reader will notice that a fixed step size of 0.3 performs best out of the fixed step size set, exactly the step size Dhaka et al. and Welandawe et al. recommend.

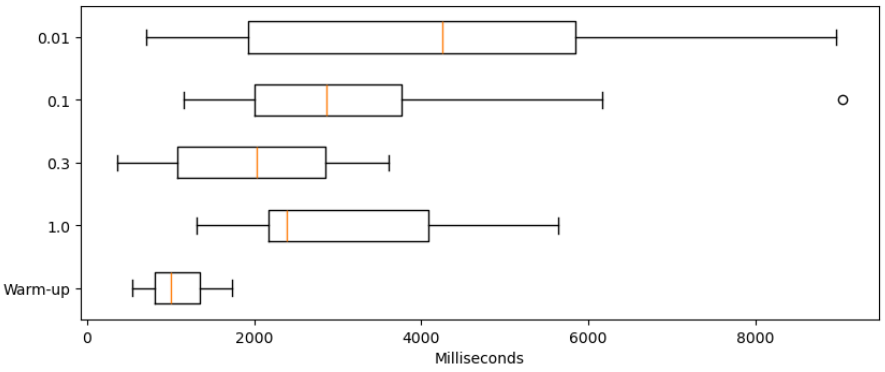

Figure 15: Runtime for warm-up versus fixed step sizes — *blr.stan*

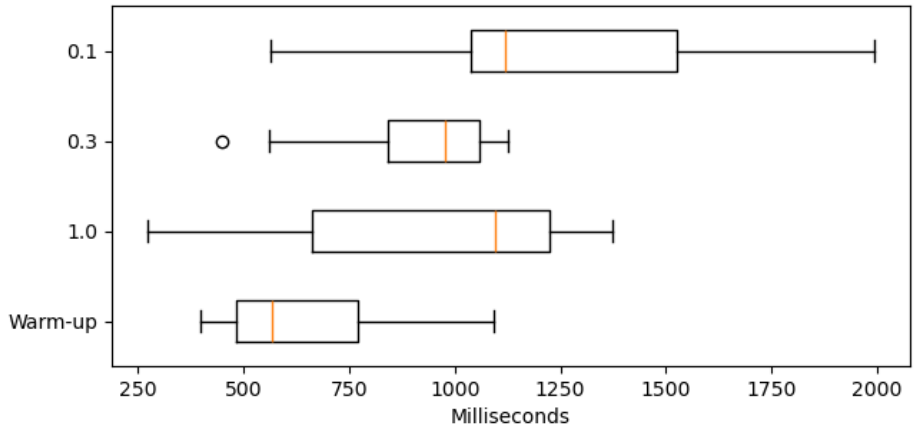

Figure 16: Runtime for warm-up versus fixed step sizes — *GLM_Binomial.stan*

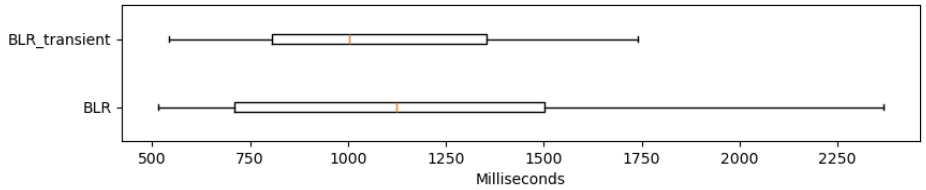

Figure 17: Runtime for warm-up with and without largest step size selection — *blr.stan*

5.3 Optimization phase

When our warm-up phase (section 5.2) is finished, we have a variational approximation and a step size to start the optimization process with. As with many numerical optimization problems, it is challenging to formulate robust termination criteria for VIs optimization [16, 25, 35, 49]. In section 4, we have shown that one of the issues with ADVI in Stan is its termination criterion. Therefore, our primary focus has been to develop a reliable and robust termination criterion that works regardless of model and data specifications.

We build upon the ideas of Pflug et al., whose work focussed on the convergence of fixed step size stochastic optimization, and the more recent work by Welandawe et al., who adapted Pflug's work to a VI setting [32, 45]. Welandawe et al. adapt Pflug's fixed step size optimization technique with their own convergence check and expand the method with a termination rule [45]. We have a similar approach for the fixed step size optimization process but a novel termination rule for the overall

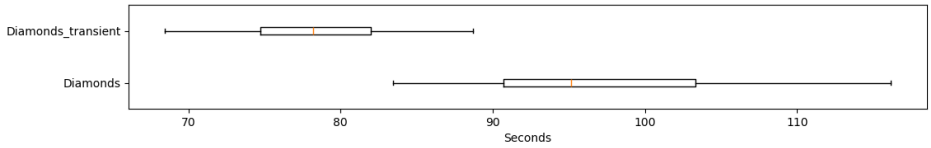

Figure 18: Runtime for warm-up with and without largest step size selection — *Diamonds.stan*

optimization process.

Like the mentioned authors, we treat the series of variational approximations produced by the optimization with a fixed step size as a Markov chain:

$$\boldsymbol{\lambda}^{\delta} \coloneqq \lambda^{(1)}, \lambda^{(2)}, \lambda^{(3)}, \ldots \tag{9}$$

Here, δ is the current step size, and the λ are produced by equation 4, i.e., the optimization process. Dhaka et al. and Welandawe et al. use the $\hat{R}$ statistic from MCMC literature to detect convergence in this Markov chain [11, 45]. Normally, $\hat{R}$ is calculated using multiple chains, i.e., parallel MCMC runs. However, we only run a single chain; see section 7 for our idea on multi-chain runs of VI. The multi-chain approach of MCMC calculates $\hat{R}$ as follows:

$$\hat{R} \coloneqq (\mathbb{V}/\mathbb{W})^{\frac{1}{2}} \tag{10}$$

Where $\mathbb{V}$ and $\mathbb{W}$ are the between-chain and within-chain variances respectively [18]. Since we only run a single chain, we use the so-called split-$\hat{R}$ version, where we split the Markov chain in half and treat each half as a separate chain [43]. A split-$\hat{R}$ less than 1.1 is a necessary but insufficient condition for stationarity. The idea of using it to detect convergence of VI was proposed by Dhaka et al. and works well in practice [11, 43].

If the $\hat{R}$ statistic - we use $\hat{R}$ and split-$\hat{R}$ interchangeably - is close to 1, it suggests that the iterates in the window W are close to stationarity [45]. split-$\hat{R}$ is formulated in terms of $\hat{R}$ from equation 10. This $\hat{R}$ is calculated for each dimension, where the between-chain variance is the variance between the first and second half of the iterates of that dimension. The maximum $\hat{R}$ over all these dimensions is the resulting split-$\hat{R}$:

$$\text{split-}\hat{R}(W) \coloneqq \max_{1 \leq i \leq d} \hat{R}(\lambda_i^{(k-W+1)}, \ldots, \lambda_i^{(k)}) \tag{11}$$

Here, k is the current iteration, W is the size of the window for look-back, and i represents the ith dimension. Dhaka et al. and Welandawe et al. suggest using split-$\hat{R} \leq 1.1$ as the threshold for detecting stationarity. While Dhaka et al. use a fixed W (default 100) we follow Welandawe's approach of searching for a window size between W_{min} and $0.95k$ to find a W that minimizes $\hat{R}$. A minimum window size is needed to ensure the robustness of the approximations, while the maximum takes into account a bit of warm-up. Welandawe et al. search over this range with 5 evenly spaced values. We adopt this exact same approach and do a robustness experiment on the minimum window size; see section 6.3.

When the optimization process is stationary according to $\hat{R}$, we must decide whether to terminate the process or continue with a smaller step size. Welandawe et al. strike a balance between possible

accuracy gained and expected runtime would they continue the process. They do this by suggesting a *symmetrized* Kullback-Leibler divergence (SKL) and a Relative accuracy Improvement statistic (RI), combining these results in an inefficiency index which they use to determine termination [45].

Estimating RI involves predicting the number of iterations the optimization requires before reaching stationarity with a smaller step size. Their prediction assumes that the number of iterations to convergence grows exponentially with a decrease in step size [45]. However, this claim is only tested on a Gaussian model with a diagonal non-identity covariance matrix: $\Sigma_{ij} = j\mathbb{1}[i = j]$. Typical Bayesian models, such as hierarchical models, will not have a diagonal covariance matrix; therefore, this argument will not generally hold. We have thus chosen not to use any prediction of future runtimes.

Welandawe et al. came up with SKL because they were looking for a measure that guarantees the accuracy of the mean, covariance, and interval probabilities. While the 2-Wasserstein distance provides this, the authors think it does not provide strong enough guarantees on the interval probabilities [45]. Therefore, the authors introduce the SKL, which measures the discrepancy between the optimal and optimized variables. Since the optimal variational approximation is unknown, Welandawe et al. compare the variational approximation of the current fixed step size phase with the variational approximation of the previous fixed step size phase. However, since we are working with a mean-field Gaussian family, the SKL will compare two Gaussians per dimension. But Gaussians are fully characterized by their mean and covariance, having well-behaved tails, meaning that changes in mean and covariance translate into predictable changes in the tail. That defeats the argument of interval probabilities, which are only relevant when comparing vastly different probability distributions [9, 4].

Convergence criterion We have decided to take an approach where only a few hyperparameters have to be set and only require the user to provide a single intuitive parameter; this parameter, like all others, has an experimental default, see section 6.3. When the optimization for a step size is stationary, according to $\hat{R}$, we half the step size and continue with a subsequent fixed step size optimization run. When we have done two or more fixed step size optimization runs, we will compare the variational approximation of the current and previous run with the 2-Wasserstein distance [26]. When the 2-Wassserstein distance is below α (default 0.001) we terminate the optimization process and return the obtained variational approximation, see section 5.3 for more details on the exact variational approximation used in these comparisons. When the 2-Wasserstein distance exceeds α, we will half the step size and start another optimization epoch.

The 2-Wasserstein distance was originally used in optimal transport theory, but it can also be used to quantify the work needed to move from one probability distribution to another [39, 26]. The 2-Wasserstein distance has a neat closed-form solution for Gaussians. For example, the distance between $\mathcal{N}(\mu_1, \Sigma_1)$ and $\mathcal{N}(\mu_2, \Sigma_2)$ is:

$$\mathcal{W}_2 \coloneqq \sqrt{||\mu_1 - \mu_2||^2 + \text{Trace}(\Sigma_1 + \Sigma_2 - 2(\Sigma_2^{\frac{1}{2}}\Sigma_2\Sigma_1^{\frac{1}{2}})^{\frac{1}{2}})} \tag{12}$$

Since we are optimizing with the mean-field Gaussian assumption, recall that we assume each model parameter is independent. Therefore, we can compare each dimension independently, and

the 2-Wasserstein distance for two Gaussians simplifies to:

$$\mathcal{W}_2 \coloneqq \sqrt{(\mu_1 - \mu_2)^2 + (\sigma_1 - \sigma_2)^2} \tag{13}$$

Our complete comparison thus becomes:

$$\bigwedge_{i=1}^{d} \left(\sqrt{(\mu_i^{(k_\delta)} - \mu_i^{(k_{\delta/2})})^2 + (\sigma_i^{(k_\delta)} - \sigma_i^{(k_{\delta/2})})^2} < \alpha \right) \tag{14}$$

Where d is the dimension of the Bayesian model we are fitting and k_δ is the variational approximation at the iteration where the optimization epoch for step size δ reached stationarity. When the equation holds true, we terminate the process and return the variational approximation to the user. The requirement for the user is to set α to a value they deem appropriate. The benefit of the 2-Wasserstein distance is that α has an intuitive meaning for the user. Since the difference in the means and standard deviations is always below α if the comparison of equation 14 is 1. Therefore, if we make the assumption that subsequent optimization epochs with a smaller step size make increasingly fewer changes to the variational approximation, then α becomes the measure of accuracy that the user wants. Consider the change in the variational approximation between epoch t and $t+1$ as $\lambda^{(t)} - \lambda^{(t+1)}$, then our assumption is that for any subsequent epoch $t+s$ it will hold that $\lambda^{(t)} - \lambda^{(t+s)} \leq \lambda^{(t)} - \lambda^{(t+1)}$.

Iterate averaging Iterate averaging, invented by Polyak et al. and later adapted for use in VI by Dhaka et al., is a technique we adopt; see section 2.3.1. Iterate averaging works by considering the series of variational parameters produced by a fixed step size optimization process as a Markov chain. When this Markov chain reaches stationarity, the average of these iterates will return a more exact solution of the true posterior than the last approximation [32, 11, 34].

$$\boldsymbol{\lambda}^\delta \coloneqq \lambda^{(1)}, \lambda^{(2)}, \lambda^{(3)}, \ldots$$

Where δ is the step size of the fixed step size optimization epoch, and $\lambda^{(i)}$ is the variational approximation after the i'th update. It is important to average over those variational approximations considered part of the stationary Markov chain, i.e., allowing for some warm-up time. Welandawe et al. take this further and use an MCSE and ESS check before averaging; when this check does not succeed, they continue the optimization process. We have chosen to drop this requirement and use the already built-in feature of window selection that is part of the stationarity check. When the fixed step size optimization halts, it means we detected stationarity through $\hat{R}$. $\hat{R}$ checks the last W variational approximations, determining W is described in section 5.3. We use this same W as the range to average over.

$$\hat{\lambda}^\delta \coloneqq \frac{1}{W+1} \sum_{i=k_\delta - W}^{k_\delta} \lambda^{(i)} \tag{15}$$

Here, k_δ is the iteration at which the optimization with fixed step size δ converged, and W is the window size determined by the stationarity check. The iterate average $\hat{\lambda}^\delta$ is what we use in the termination rule to determine if the process has converged, see section 5.3.

5.4 Diagnostics

Computing a variational approximation of the true posterior is always possible. However, determining if the approximation is (near) optimal is difficult [47, 44]. A similar difficulty exists with evaluating the posterior computed by MCMC algorithms, where the $\hat{R}$ statistic is widely used to alert of problems [43]. For variational inference, the k-hat statistic has a comparable role to $\hat{R}$ for MCMC; see section 2.3.1 for an explanation of k-hat. In short, after computing a variational approximation, we take draws from this approximation and the joint density to calculate importance weights (IW). From the largest 20% of IW a generalized Pareto distribution is fit. This fit produces the shape parameter, k, similar to a μ for a Gaussian distribution. Vehtari et al. showed empirically that a shape parameter below 0.7 indicates a usable variational approximation.

We have chosen to incorporate the k-hat diagnostic and provide the user with this diagnostic. Thus providing some indication of the reliability of the variational approximation. Merging our variational inference algorithm with Stan would allow the k-hat diagnostic to be supplied like the $\hat{R}$ is provided for MCMC-produced densities now. However, experiments with k-hat (section 6.2) demonstrate their ineffectiveness.

5.5 Complete algorithm

Combining the warm-up and optimization phases from sections 5.2 and 5.3 yields our complete algorithm. Note that initialization through prior distributions is left out due to implementation difficulties in Stan. This section provides the algorithm in pseudo-code to provide a clear picture while hiding implementation details. In algorithm 1, we summarize the warm-up phase, and in algorithm 2, the optimization part. The complete algorithm has five parameters, four of which are hyperparameters, and have verified defaults found through a robustness test, see section 6.3. Only a single parameter (the accuracy, α) must be set by the user, which can also be omitted because a generally good default is provided. This single parameter, as described in section 2, exists on an easy-to-interpret scale for an end user:

- **accuracy** α: The VI algorithm compares, whenever it reaches stationarity, the Gaussian approximation with the previous one and halts optimization when the change per dimension is below α. Assuming that subsequent changes, if we continue optimizing, will be less than α this parameter can be considered an accuracy measure. This makes it easy to interpret, considering that the end user knows the level of accuracy necessary for their model.
- **window** W (default: 200): This default value has been found empirically and is also suggested by Welandawe et al. Increasing it will provide a bit more accuracy on a good portion of the models but will increase the runtimes significantly. A user might benefit by increasing this value when their posterior is not accurate enough, and they have enough time.
- **Monte Carlo samples** M (default: 15): Experiments show that around 15 gradient samples result in robust gradient calculations while not being too much of a computational cost. The user can always increase this number for more accuracy, but we suspect no large gains will be made. Providing a value lower than 10 will cause large oscillations in gradient calculations.
- **minimum step size** δ_{min} (default: 10^{-5}): We find that having smaller step sizes leads to excessive computation times while not improving the approximation.

- **maximum iterations** k_{max} (default: 100 000): The maximum iterations must always be set, and a warning is returned when it is reached before convergence. This allows the user to act upon that info by changing the model or increasing the maximum iterations.

When running our algorithm, we always start with the warm-up phase; we can see from algorithm 2 that algorithm 1 is called. Algorithm 1 is the warm-up algorithm and starts by defining a list of step sizes ranging from 10^3 to 10^{-5} on line 2. With this list, a function `BestStepSize()` is called, which, for each step size in the provided list, runs an optimization round for the provided window length (lines 34-42). Subsequently, it returns the step size and the variational approximation for the step size that had the largest ELBO. This step size and variational approximation are returned and stored (line 3). Then, with that step size, we make a new list of step sizes, one an order of magnitude smaller and one larger (line 9). With this list, the function `BestStepSize()` is run again, and the result is stored (line 10). We keep doing this process until we run the optimization three times, but not increasing the ELBO faster than two standard deviations of the previously found ELBO (lines 12-17). In the next loop (lines 20-30) we find the largest step size that does not have an ELBO more than two standard deviations from the thus far largest ELBO. We do this by running the optimization for 100 runs with the step size that we previously doubled. If its ELBO is within two standard deviations, we double again until this does not hold. Returning in the end, the largest step size that still had an ELBO within two standard deviations, the variational approximation up to this point, and the total iterations ran (line 32).

After the warm-up phase, the optimization phase starts, which is described in algorithm 2. On line 1, we can see the call to algorithm 1. The total iterations up to this point is stored as the iterations since the optimization was last stationary (line 2). Now we enter a loop until the optimization process converges or reaches the maximum number of iterations, the latter is left out for brevity (lines 4-27). For each loop, the optimization is run for W iterations with M Monte Carlo samples (line 5). Subsequently, the split-$\hat{R}$ diagnostic was calculated on five window lengths across the last k iterations since stationarity (lines 7-10). If the split-$\hat{R}$ is larger than 1.1, the loop continues to the next iteration, where we again run the optimization (line 12). When split-$\hat{R}$ is less than or equal to 1.1, we calculate the variational approximation by averaging over the last w approximations, where w is the window over which split-$\hat{R}$ is calculated (line 13). Next, the 2-Wasserstein distance is computed using the average variational approximation compared to the one from the last time we were stationary (line 14). If this 2-Wasserstein is smaller than the provided α, the optimization is terminated, and we proceed to line 29 (lines 15-18). If the current step size is greater than the minimum step size provided, we half the current one (lines 20-22). Next, we update the iterations since stationarity to the current total iterations and update the variational approximation of the current step size to that of the iterate average (lines 24-25). When the loop terminates due to convergence or reaching the maximum number of iterations, we compute the k-hat diagnostic on the current iterate average variational approximation and return that approximation with its diagnostic (lines 29-30).

In summary, the warm-up phase picks an appropriate step size for the model and tries to make progress quickly. After, we find the largest step size, which does not destabilize the optimization process. We start the optimization phase with this step size, checking for stationarity every W iterations. In the case of stationarity, we compare the current parameters with the parameters of the previous time we were stationary. We declare the optimization converged if these differences are below a threshold α. If the differences are larger than α, we decrease the step size and continue the optimization. After convergence, we diagnose our posterior with the k-hat diagnostic and provide

it, together with the variational approximation, back to the user.

Algorithm 1: Warm-up & transient phase

Input: window size W (default: 200), gradient iterations M (default: 15)
Output: k, λ, δ

$t = 0$
step_sizes $\leftarrow [1000.0, 100.0, 10.0, 1.0, 0.1, 0.01, 0.001, 0.0001, 0.00001]$
δ_t, λ_t, elbo$_t$, $\sigma_t \leftarrow$ `BestStepSize(`*NULL, step_sizes,* W`)`
$k \leftarrow W$ `// After BestStepSize() we ran` W `iterations`

count $\leftarrow 0$
while *True* **do**
  $t \leftarrow t + 1$
  step_sizes $\leftarrow [\delta_{t-1} * 2, \delta_{t-1}, \delta_{t-1}/2]$
  δ_t, λ_t, elbo$_t$, $\sigma_t \leftarrow$ `BestStepSize(`λ_{t-1}, *step_sizes,* W`)`
  $k \leftarrow k + W$
  if $elbo_t < elbo_{t-1} + 2 * \sigma_{t-1}$ **then**
    count = count + 1
    if *count* $== 3$ **then**
      break
    end
  end
end

while $t \leftarrow t + 1$ **do**
  $\lambda_t \leftarrow$ optimization(λ_{t-1}, $\delta_{t-1}, 100, M$)
  $k \leftarrow k + 100$
  elbo$_t$, $\sigma_t \leftarrow$ calculate_elbo(λ_t)
  if $elbo_t > elbo_{t-1} + 2 * \sigma_{t-1}$ ***or*** $elbo_t < elbo_{t-1} - 2 * \sigma_{t-1}$ **then**
    $\delta_t \leftarrow \delta_t/2$
    break
  else
    $\delta_t \leftarrow \delta_t * 2$
  end
end

return k, λ_t, δ_t

Function `BestStepSize(`λ_{t-1}, *step_sizes,* W`):`
  $\lambda \leftarrow$ NULL `// best variational approximation`
  $\delta \leftarrow 0.0$ `// best step size`
  $ELBO \leftarrow -\infty$ `// best ELBO`
  foreach *ss in step_sizes* **do**
    $\lambda_{ss} \leftarrow$ optimization(λ_{t-1}, ss, W, M)
    `// sigma is the square root of the variance`
    elbo, $\sigma \leftarrow$ calculate_elbo(λ_{ss})
    if $elbo > ELBO$ **then**
      $\lambda \leftarrow \lambda_{ss}$
      $\delta \leftarrow ss$
      ELBO $\leftarrow$ elbo
    end
  end
  return δ, λ, elbo, σ
end

Algorithm 2: Optimization phase

Input: accuracy α (default: 0.001), window size W (default: 200), Monte Carlo samples M (default: 15), minimum step size δ_{min} (default: 10^{-5})

Output: λ, $\hat{k}$

$k, \lambda_k, \delta \leftarrow$ `WarmUp(`W, M`)` `// Call to algorithm 1`

$k\_since\_stationarity \leftarrow k$

converged $\leftarrow False$

while *not* *converged* **do**

  $\lambda_{k+W} \leftarrow$ optimization$(\lambda_k, \delta, W, M)$

    `// A check is done to ensure at least` W `iterations were ran in warm-up`

  $R_\delta \leftarrow (k\_since\_stationarity * 0.95 - W)/4$

  $R_{\text{windows}} \leftarrow [W, W + R_\delta, W + 2R_\delta, W + 3R_\delta, W + 4R_\delta]$

  $R_{\text{hats}} \leftarrow [\text{r\_hat}(w)$ for w in $R_{\text{windows}}]$ `// Call Stan's r_hat function`

  $\hat{R}, w \leftarrow min(R_{\text{hats}})$ `//` $\hat{R}$ `and window for that smallest` $\hat{R}$

  if $\hat{R} \leq 1.1$ **then**

    $\lambda_{\hat{R}} \leftarrow$ `IterateAveraging(`w`)` `// Per dimension`

    $\xi \leftarrow$ `2WassersteinDistance(`$\lambda_{\hat{R}}, \lambda_{k\_since\_stationarity}$`)`

    if $\xi < \alpha$ **then**

      converged $\leftarrow True$

      break

    end

    if $\delta > \delta_{min}$ **then**

      $\delta \leftarrow \delta/2$

    end

    $k\_since\_stationarity \leftarrow k$

    $\lambda_{k\_since\_stationarity} \leftarrow \lambda_{\hat{R}}$

  end

end

$\hat{k} \leftarrow$ `KHat(`$\lambda_{\hat{R}}$`)`

return $\lambda_{\hat{R}}$, $\hat{k}$

6 Experiments

This section describes the experiments and the results thereof. Here, we compare our algorithm as defined in section 5.5 with Stan's MCMC, specifically NUTS. We run our experiments on a laptop with the following specifications [12]:

- **Processor**: Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz (8 CPUs)
- **RAM**: 16 GB DDR3 2133 MHz
- **OS**: Ubuntu 22.04.3 LTS
- **Intel Turbo Boost**: Disabled

To compare our algorithm with MCMC, we use PosteriorDB [40]. PosteriorDB is a set of models with datasets, implementations in Stan, and reference posteriors in the form of samples of that posterior. Of the total 123 posteriors, we picked 16 that are representative and cover many model types. The models for which a reference posterior exists have a set of 100 000 roughly independent samples. For the remaining models, for which no reference posterior exists, we ran MCMC with identical settings that PosteriorDB uses for their reference posteriors:

- **Stan version**: v2.33.0 (5 September 2023)
- **chains**: 10
- **iter**: 20 000
- **warmup**: 10 000
- **thin**: 10
- **ndraws**: 10 000

PosteriorDB requires their reference posteriors to have zero divergent transitions. However, while running MCMC for the models without reference posteriors, we found that divergent transitions occurred frequently, which might explain their absence in PosteriorDB. Therefore, we have chosen only to accept draws for runs with a maximum of 1 divergent transition per 1000 iterations. This number of divergent transitions can still be problematic, but we consider this a reasonable tolerance for our use case.

The set of models from PosteriorDB we use to do our experiments includes:

- Hierarchical models: `dogs, dogs_hierarchical, eight_schools_centered, surgical`
- Hidden Markov models: `hmm_example`
- Time series models: `arK, garch11`
- Linear models: `GLM_Binomial, sblrc, sblri, wells_dist`
- Non-Linear models: `dugongs`
- Log-Log models: `diamonds`

[12] Due to the weaker specs compared to newer systems, the experiments, such as runtime analysis, might be representative of what an end-user might expect.

- Mixture models: `low_dim_gauss_mix`
- Gaussian process models: `gp_pois_regr`
- Mark-Recapture models: `M0_model`

For each of these models, we run our VI algorithm 100 times with its default settings:

- **accuracy** α: 0.001
- **window** W: 200
- **Monte Carlo samples** M: 15
- **minimum step size** δ_{min}: 10^{-5}
- **maximum iterations** k_{max}: 1 000 000

We have 100.000 samples from the posterior, fitted by MCMC, for each model. We calculate the mean and standard deviation from these samples per dimension. Since our VI uses mean-field Gaussians to approximate the posterior, we use the Gaussians' mean and standard deviation to compare to MCMCs rather than sampling from the posterior. The tables below compare these descriptive statistics per model and dimension; if the VI mean is not within two standard deviations of the MCMC mean, we set the row in bold.

Furthermore, we provide a table with summary statistics for each model. We calculate the standardized mean difference per dimension, i.e., how many standard deviations our VI mean deviates from the MCMC mean. For this list of standardized mean differences, we compute the Root Mean Square Error, the minimum, and the maximum and show these per model in a single plot.

From tables 2 through 17, we see that the models `dogs_hierarchical, dugongs, garch11, hmm_example, low_dim_gauss_mix, M0_model` have means of one or more dimensions that deviate at least two standard deviations from their MCMC counterpart. We do not have a clear-cut answer as to why VI performs poorly on these models. However, we presume it is due to the hierarchical nature (`dogs_hierarchical, hmm_example`, multimodality (`M0_model, low_dim_gauss_mix`), and autocorrelation (`garch11`). Although VI has a hard time on these models, it performs well across the board on various model types. In figure 19 of the standardized mean differences per dimension, the RMSE, minimum and maximum values are plotted. From this, we can deduce that six models perform worse than the two standard deviation tolerances. In contrast, the remaining models perform very well, with most having no dimension with more than 10^{-1} deviation and none with more than 10^{0} deviation. This shows our VI algorithm provides parameters that are accurate and reliable enough for inference in most cases.

dogs-dogs			
Mean		Std Dev	
VI	MCMC	VI	MCMC
1.800	1.809	0.104	0.230
−0.359	−0.359	0.026	0.037
−0.210	−0.211	0.018	0.043

Table 2

dogs-dogs_hierarchical			
Mean		Std Dev	
VI	MCMC	VI	MCMC
2.495	**0.923**	**0.129**	**0.011**
1.298	**0.786**	**0.094**	**0.019**

Table 3

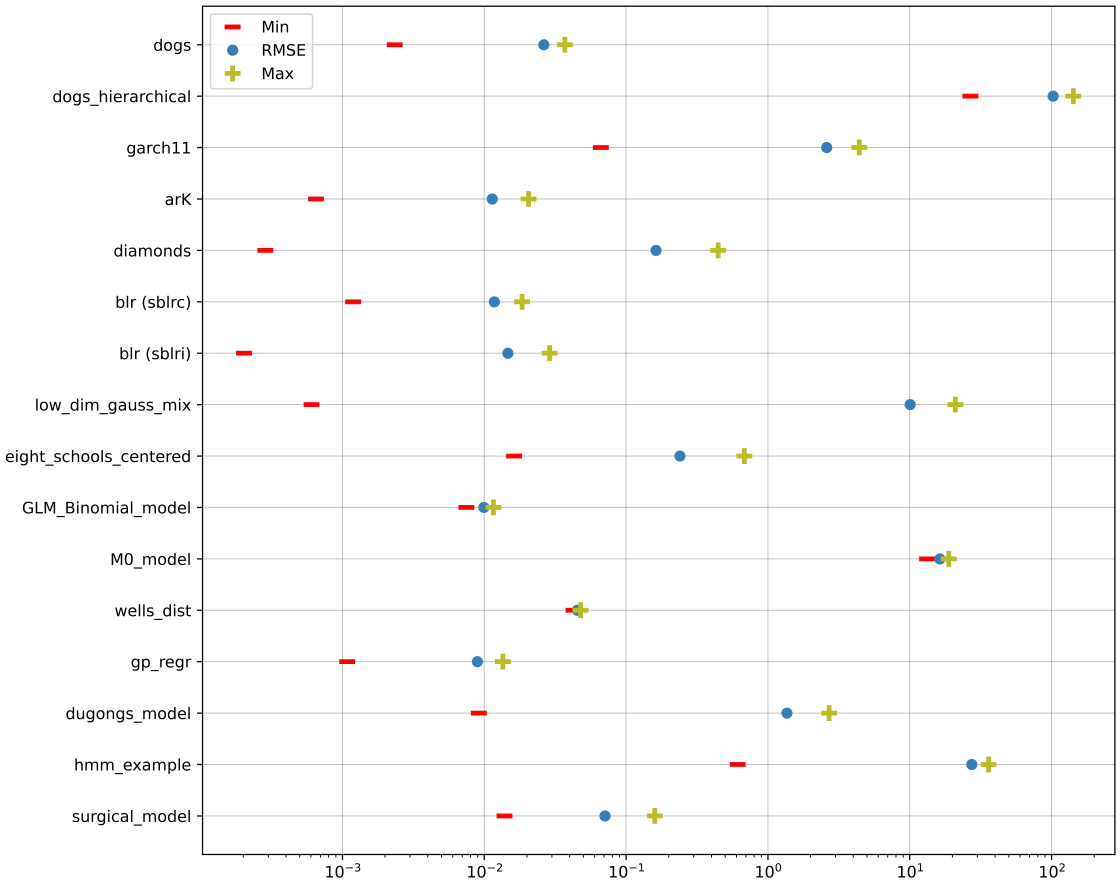

Figure 19: RMSE, Min and Max value of the standardized mean difference per dimension. The mean and standard deviation of MCMC are used as the golden standard.

garch-garch11			
Mean		Std Dev	
VI	MCMC	VI	MCMC
5.058	5.050	0.117	0.123
0.342	1.475	0.202	0.569
0.330	0.569	0.461	0.126
0.834	**0.291**	**0.563**	**0.123**

Table 4

arK-arK			
Mean		Std Dev	
VI	MCMC	VI	MCMC
−0.001	−0.001	0.011	0.011
0.691	0.692	0.021	0.071
0.440	0.440	0.021	0.087
0.105	0.105	0.021	0.092
−0.036	−0.036	0.021	0.086
−0.301	−0.301	0.021	0.069
0.151	0.150	0.008	0.008

Table 5

diamonds-diamonds			
Mean		Std Dev	
VI	MCMC	VI	MCMC
6.645	6.660	0.004	0.251
6.244	6.363	0.012	0.328
−4.560	−4.684	0.012	0.321
1.450	1.447	0.009	0.141
0.134	0.134	0.005	0.007
−0.040	−0.040	0.004	0.007
0.023	0.023	0.004	0.006
0.002	0.002	0.004	0.004
−0.445	−0.445	0.005	0.006
−0.093	−0.093	0.005	0.006
−0.013	−0.013	0.005	0.005
0.011	0.011	0.004	0.005
−0.001	−0.001	0.004	0.005
0.001	0.001	0.004	0.004

Table 6

sblrc-blr			
Mean		Std Dev	
VI	MCMC	VI	MCMC
1.000	1.000	0.001	0.001
0.999	0.999	0.001	0.001
0.998	0.998	0.001	0.001
0.999	0.999	0.001	0.001
0.999	0.999	0.000	0.001
1.041	1.042	0.004	0.077

Table 7

sblri-blr			
Mean		Std Dev	
VI	MCMC	VI	MCMC
0.999	0.999	0.001	0.001
1.000	1.000	0.001	0.001
1.000	1.000	0.001	0.001
1.001	1.001	0.001	0.001
1.002	1.002	0.001	0.001
0.961	0.963	0.004	0.071

Table 8

low_dim_gauss_mix-low_dim_gauss_mix			
Mean		Std Dev	
VI	MCMC	VI	MCMC
−2.733	−2.734	0.033	0.042
1.723	**2.870**	**0.009**	**0.055**
1.028	1.028	0.029	0.031
1.024	1.024	0.039	0.040
0.496	**0.622**	**0.066**	**0.015**

Table 9

eight_schools-eight_schools_centered			
Mean		Std Dev	
VI	MCMC	VI	MCMC
7.127	6.507	5.242	5.649
5.171	5.080	4.798	4.704
3.451	3.959	5.278	5.350
4.758	4.876	4.928	4.839
2.797	3.619	4.668	4.722
3.670	4.156	4.907	4.908
7.619	6.573	4.876	5.112
4.877	4.965	5.322	5.391
4.279	4.493	1.825	3.227
1.772	3.931	0.241	3.164

Table 10

GLM_Binomial			
Mean		Std Dev	
VI	MCMC	VI	MCMC
1.015	1.014	0.041	0.065
−0.173	−0.173	0.073	0.073
−0.921	−0.922	0.096	0.146

Table 11

M0_model			
Mean		Std Dev	
VI	MCMC	VI	MCMC
−0.311	**0.423**	**0.148**	**0.039**
−0.009	**0.497**	**0.137**	**0.038**

Table 12

wells_dist			
Mean		Std Dev	
VI	MCMC	VI	MCMC
0.610	0.607	0.037	0.061
−0.006	−0.006	0.002	0.001

Table 13

gp_pois_regr-gp_regr			
Mean		Std Dev	
VI	MCMC	VI	MCMC
6.884	6.902	1.306	1.300
2.440	2.434	0.789	0.789
1.832	1.833	0.512	0.500

Table 14

dugongs			
Mean		Std Dev	
VI	MCMC	VI	MCMC
2.636	2.654	0.018	0.073
0.959	0.973	0.045	0.077
0.951	**0.863**	**0.092**	**0.033**
109.168	109.461	30.272	31.962

Table 15

hmm_example			
Mean		Std Dev	
VI	MCMC	VI	MCMC
0.727	0.666	0.472	0.100
−2.612	**0.334**	**0.430**	**0.100**
1.106	**0.073**	**0.032**	**0.029**
1.758	**0.927**	**0.019**	**0.029**

Table 16

surgical			
Mean		Std Dev	
VI	MCMC	VI	MCMC
−2.551	−2.553	0.115	0.152
0.189	0.186	0.081	0.160
−2.945	−2.952	0.341	0.435
−2.158	−2.184	0.221	0.235
−2.610	−2.605	0.265	0.273
−2.785	−2.774	0.139	0.144
−2.968	−2.942	0.247	0.280
−2.627	−2.622	0.230	0.232
−2.675	−2.667	0.259	0.263
−1.948	−1.981	0.182	0.208
−2.618	−2.607	0.227	0.228
−2.500	−2.496	0.276	0.276
−2.167	−2.187	0.184	0.192
−2.630	−2.624	0.185	0.185

Table 17

6.1 Runtime analysis

One major selling point of VI with respect to MCMC is its supposed speed [49, 5]. In figure 20, we compare our VI algorithm runtimes with MCMC runtimes for 100 runs each. We run MCMC with its default settings of 4 chains and 2000 iterations, including 1000 warmup iterations. The times for the 100 runs of VI and MCMC are plotted in a distinct boxplot on the row for the corresponding model. Where the box spreads from the first to third quartile with a line at the median. The whiskers extend from two sides to the farthest point, lying within 1.5 times the interquartile range; data points outside this range are shown as circles [1].

The assumed speed difference would be most notable when there is a large dataset since VI can use batching to speed up the optimization. However, our algorithm is built in Stan and has no way to run on subsets of the data. Also, there are not many large datasets in PosteriorDB, except for neural net models, and we have thus not compared with any large model with a corresponding large dataset. In the runtime plot, figure 20, we can see that MCMC beats our VI algorithm on almost all models. Closer inspection reveals that VI is faster on the `diamonds` and `arK` model. Both have a large number of dimensions compared to all other models we included in our analysis. More analysis on runtimes corrected for dimensions must be done to provide a verdict. Still, longer runtimes in higher dimensions make intuitive sense for MCMC due to the lower acceptance rate at higher dimensions.

Furthermore, we are using a VI algorithm that must still be considered a prototype and has not been optimized due to time constraints. Since the runtimes of VI are always within an order of magnitude (except in the two cases where no convergence was reached, but the optimization halted due to reaching the maximum iterations) of the MCMC runtimes, we presume that we can optimize this discrepancy out.

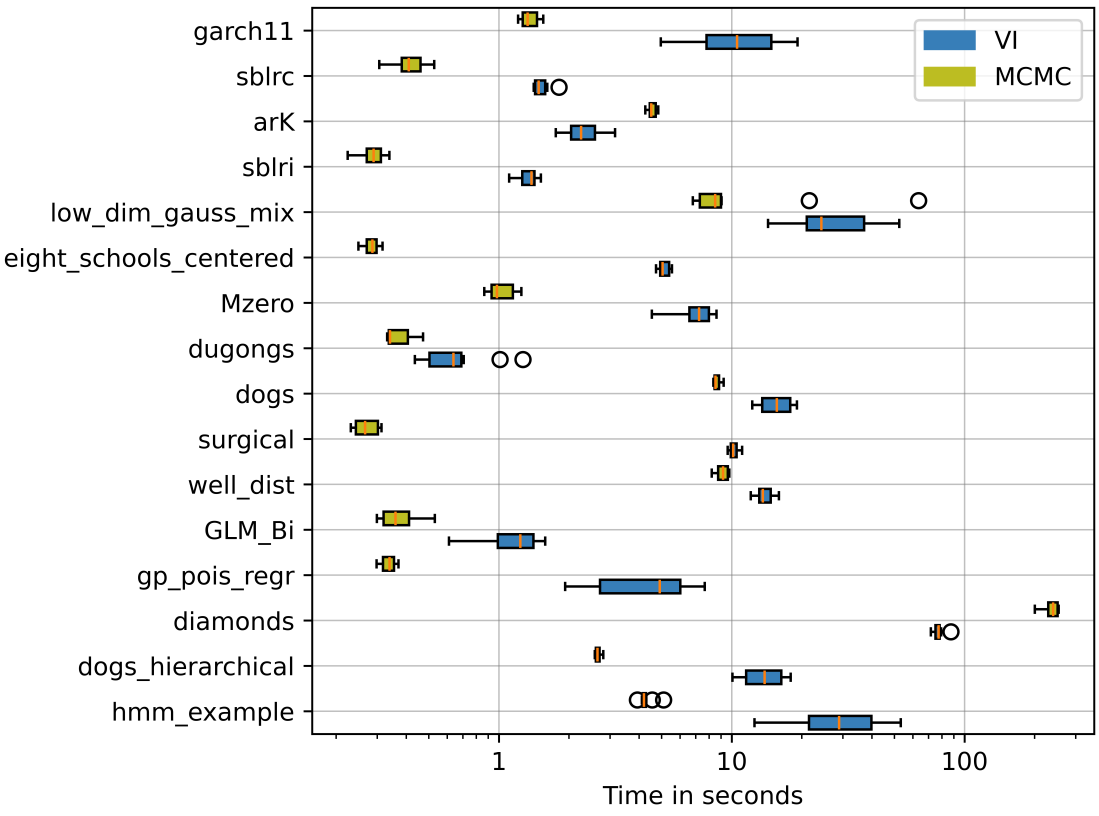

Figure 20: Runtime of VI vs. MCMC on the same model with default settings for 100 runs.

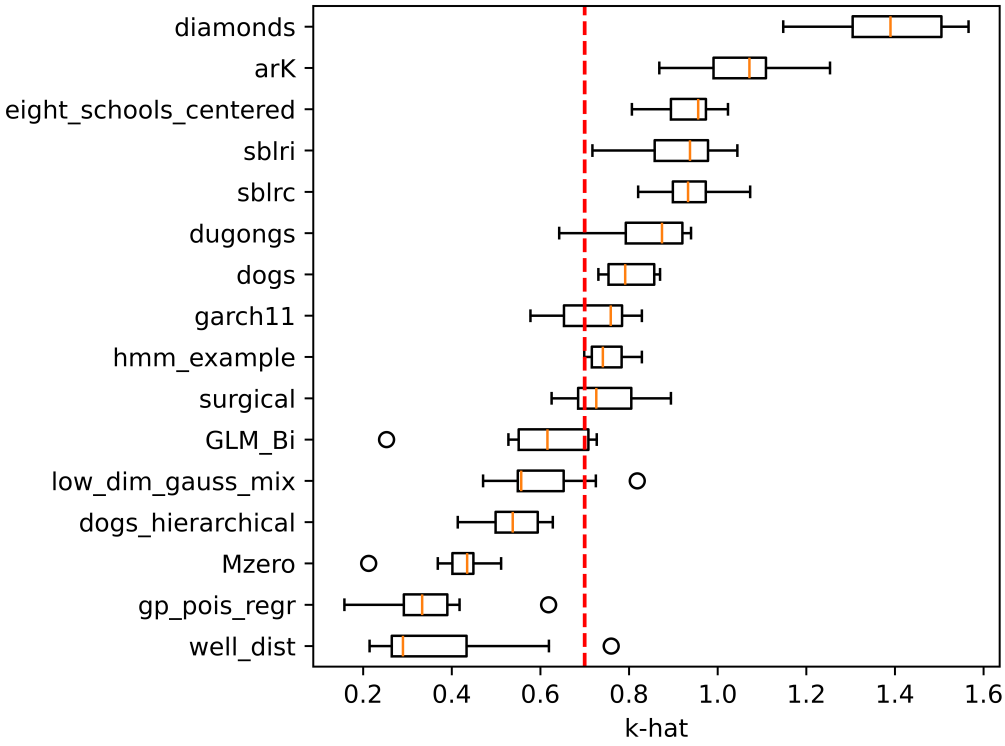

Figure 21: k-hat of VI posterior for 100 runs per model. A vertical striped line at the cut-off 0.7 as provided by Vethari et al.

6.2 k-hat diagnostic efficacy

If the fit is not close to that of the true posterior, in this case MCMC, we want to signal this back to the end user. One diagnostic, as suggested in section 2.3.1, was k-hat, which we implemented for our VI algorithm. We ran our VI algorithm for 100 runs per model and calculated k-hat based on the largest 400 out of 2000 samples, as suggested in the paper on k-hat [44]. We plot the 100 k-hat values as a boxplot with the same settings as the runtime plot.

From the plot, figure 21, we can see only three models have had zero runs with no k-hat above 0.7. According to our earlier analysis, two of these three have the worst performance compared to MCMC. Considering the six models with most of their runs below a 0.7 k-hat, we can see that two are multimodal and four are the worst-performing models compared to MCMC. Also, both the models sblri and sblrc have all runs above 0.7, but the calculated parameters are identical to those of MCMC, while both are simple linear regression models. These observations make us skeptical about the accuracy and, thus, the efficacy of k-hat as a diagnostic.

6.3 Robustness

Our VI algorithm has four hyperparameters and a single parameter the user must set. We ran experiments with a subset of our chosen models to provide useful defaults (bold) and gauge the robustness of different choices of these parameters. Specifically, with the models `arK, dogs, sblrc, surgical, wells_dist`.

- accuracy α: $0.1, 0.01, \mathbf{0.001}, 0.0001$
- window W: $100, \mathbf{200}, 300, 500, 1000$
- Monte Carlo samples M: $10, \mathbf{15}, 20$
- minimum step size δ_{min}: $\mathbf{10^{-5}}, 10^{-6}, 10^{-7}$
- maximum iterations k_{max}: $10^5, \mathbf{10^6}, 10^7$

The default for the accuracy parameter is hand-picked since any parameter will work fine for the accuracy level one wants. Providing 0.1 for α ensures all parameters differed less than 0.1 between stationary points. We found that setting α to 0.001 gave our chosen models the best accuracy compared to MCMC. However, mileage may vary with different models. For finding the minimum step size and maximum iteration parameter, we used the defaults for *window* and *Monte Carlo samples* provided by Welandawe et al. [45], 200 and 10, respectively. This gave us only nine combinations of parameters to search over. Once finished, we used the defaults found for *minimum step size* and *maximum iterations* to run the same experiment for *window* and *Monte Carlo samples*, only resulting in a search space of 15 combinations.

After finding default values for all parameters, we ran the robustness experiment. We use the same setup as Welandawe et al.: We vary one parameter at a time and keep the defaults for the others. From these runs, we can conclude that the sensitivity to parameter changes is minimal for the accuracy and runtime of our algorithm — a verdict similar to Welandawe et al.

7 Conclusion

We have developed a fully automatic, robust, and accurate algorithm that runs only with a single intuitive parameter from the user. Experiments show that it performs well on various models from PosteriorDB, and its accuracy and runtime are insensitive to changes in its hyperparameters. Our approach uses the classical mean-field Gaussian as an approximating family and the reparameterization trick to derive a gradient estimator. Our algorithm contains two phases, warm-up and optimization, which are more effective than only having an optimization phase. In the warm-up phase, we optimize until the optimization process is roughly where it needs to be. Then, the optimization picks up and continues this process by iteratively decreasing the update size until convergence is reached.

Furthermore, we have shown that including the priors provided in the model in the initialization process can result in significant optimization speed-ups. Sometimes, this causes a slight slowdown when the true posterior is near our original initialization point instead of near the priors. Implementation of this has not been possible since we can not access priors in Stan.

Converging is assessed using the closed-form 2-Wasserstein distance between two variational approximations in the same optimization trace. Using the 2-Wasserstein distance with mean-field Gaussians means we only require a threshold for this distance to claim convergence. Some assumptions about the 2-Wasserstein distance make this threshold intuitive for an end user. Comparing this to, for example, RAABBVI, a user must provide three parameter values. Most of these are not intuitive for a typical end-user of Stan but require extensive knowledge of the underlying algorithm.[45].

Stationarity of the optimization process is determined with the split-$\hat{R}$ diagnostic, a novel use introduced by Dhaka et al. [11]. When stationarity is reached, the variational approximation history has Markov Chain properties, allowing us to average over the chain to obtain a better approximation. We provide an end-user with these averaged approximations, which are more accurate posteriors.

Since we can not always trust our approximation, we want to inform the user if we detect it. Bad posteriors happen when, for example, the true posterior is hard to capture with a mean-field Gaussian. This is the case when dealing with multimodal posteriors. One possible solution we found was the Pareto Smoothed Importance Sampling shape parameter k-hat, which could be used to detect untrustworthy posteriors [44, 47]. However, as shown in section 6.2, k-hat is unreliable when diagnosing posteriors approximated by our algorithm. It sometimes lets multimodal models get through, and simple linear models will sometimes not pass. However, its biggest issue is the sheer number of models it flagged as faulty. While the approximated parameters are identical to those of MCMC. This, in conjunction with the fact it happens on linear regression models, suggests the problem is with k-hat and not VI or MCMC.

Using MCMC as a golden standard is not perfect. Since MCMC itself can provide faulty posteriors due to, for example, multimodal posteriors, we can not trust MCMC's parameters on all models. Since we are also not convinced by k-hat as a diagnostics, we are sometimes flying in the dark when it comes to knowing wether MCMC's parameters are valid.

For evaluating our algorithm we used a well-known set of Bayesian statistical models, namely PosteriorDB. PosteriorDB is used by many recent papers on new Bayesian inference algorithms

[23, 45, 11]. It is probably the best database of models and datasets for comparing two inference algorithms. However, it is not perfect due to two drawbacks. First, there are very few models with high dimensions, except for neural network models which are multimodal and thus not suitable for fitting with VI. This makes it impossible to perform analysis where we correct for a model's dimensionality. Secondly, there are very few models with lots of data, except for neural network models. This is a significant drawback for the central point of VI, namely that it utilizes data subsampling to speed up. However, since no large datasets exist, the speed-ups will be negligible compared to the overhead. Due to Stan's implementation, we were unable to do data subsampling in our algorithm.

In conclusion, our algorithm provides a robust and user-friendly VI algorithm in Stan that runs by providing a single accuracy parameter. In contrast to the current VI implementation in Stan, ADVI, which requires many parameters and is unstable or does not converge in most cases. We have shown that a reliable VI algorithm is possible. Furthermore, diagnostics are needed, but k-hat is likely not effective enough as a diagnostic to detect faulty approximations.

Future work Although we have provided some new insights into the field of VI, there are many new areas to research. We will list these in no particular order:

- As mentioned, we have tested using priors in our initialization phase by hand since this is not possible in Stan without major rewrites. However, future research could tackle the initialization with priors by either rewriting Stan or doing it in a different probabilistic programming language like Pyro.
- Instead of using Adam we should test if regular SGD, thus without momentum, can give us bounds on runtime versus accuracy. This is what Dhaka et al. and Welandawe et al. tried to accomplish with RABBVI and RAABBVI respectively [11, 45]. However, they did not use regular SGD. Intuitively, plain SGD might allow for better predictions of runtime since it uses a fixed step size.
- Pathfinder by Zhang et al. describes a multi-path pathfinder algorithm that runs a single-path pathfinder multiple times in parallel. This can cause different runs to end up in distinct locations of the parameter space. Zhang et al. then combine all single-path pathfinder runs per dimension as a mixture of Gaussians, each with equal weight. Whether this works at all is an open question, but it is an experiment that can also be easily done with our VI algorithm.
- Data subsampling is the big promise of VI, but it is not possible in Stan. Providing this would require changes to the compiler written in OCaml. While these are more practical concerns, it will grant more research opportunities.
- Multi-chain diagnostics in VI is a novel idea that can be used to diagnose bad posterior approximations by our algorithm. We would, like MCMC does, run four separate instances of our algorithm and, after, conduct the $\hat{R}$ check on these runs. Then, if they end up in separate places, $\hat{R}$ will signal this. This would result in a comparable $\hat{R}$ statistic as used by MCMC and immediately useful. We would still need to experiment with an allowable $\hat{R}$ threshold for VI.
- Welandawe et al. mention that the autocorrelation parameter in the MCSE calculation can be used as a diagnostics. This would be an alternative to k-hat and is thus worth researching.

A Multimodality image generation

Code for generating the image about a Gaussian approximation of a multimodal posterior as used in chapter 2.3.2:

```
x = np.linspace(-5, 5, 400)
gaussian = 0.35 * np.exp(-0.5 * (x - 1)**2)
multimodal = 0.25 * np.exp(-0.5 * (x + 3)**2) + 0.3 * np.exp(-0.5 * (x - 1)**2)

plt.plot(x, gaussian, color='blue')
plt.plot(x, multimodal, color='red')
plt.xlabel('Model Parameter')
plt.ylabel('Probability')
plt.ylim(0.0, 0.5)
plt.yticks(np.arange(0, 0.51, 0.05))
plt.grid(True)
plt.show()
```

B Ethics and Privacy Quick Scan

Section 1. Research projects involving human participants

P1. Does your project involve human participants? This includes for example use of observation, (online) surveys, interviews, tests, focus groups, and workshops where human participants provide information or data to inform the research. If you are only using existing data sets or publicly available data (e.g. from Twitter, Reddit) without directly recruiting participants, please answer no.

- No

Section 2. Data protection, handling, and storage

The General Data Protection Regulation imposes several obligations for the use of **personal data** (defined as any information relating to an identified or identifiable living person) or including the use of personal data in research.

D1. Are you gathering or using personal data (defined as any information relating to an identified or identifiable living person)?

- No

Section 3. Research that may cause harm

Research may cause harm to participants, researchers, the university, or society. This includes when technology has dual-use, and you investigate an innocent use, but your results could be used by others in a harmful way. If you are unsure regarding possible harm to the university or society, please discuss your concerns with the Research Support Office.

H1. Does your project give rise to a realistic risk to the national security of any country?

- No

H2. Does your project give rise to a realistic risk of aiding human rights abuses in any country?

- No

H3. Does your project (and its data) give rise to a realistic risk of damaging the University's reputation? (E.g., bad press coverage, public protest.)

- No

H4. Does your project (and in particular its data) give rise to an increased risk of attack (cyber- or otherwise) against the University? (E.g., from pressure groups.)

- No

H5. Is the data likely to contain material that is indecent, offensive, defamatory, threatening, discriminatory, or extremist?

- No

H6. Does your project give rise to a realistic risk of harm to the researchers?

- No

H7. Is there a realistic risk of any participant experiencing physical or psychological harm or discomfort?

- No

H8. Is there a realistic risk of any participant experiencing a detriment to their interests as a result of participation?

- No

H9. Is there a realistic risk of other types of negative externalities?

- No

Section 4. Conflicts of interest

C1. Is there any potential conflict of interest (e.g. between research funder and researchers or participants and researchers) that may potentially affect the research outcome or the dissemination of research findings?

- No

C2. Is there a direct hierarchical relationship between researchers and participants?

- No

Section 5. Your information.

This last section collects data about you and your project so that we can register that you completed the Ethics and Privacy Quick Scan, sent you (and your supervisor/course coordinator) a summary of what you filled out, and follow up where a fuller ethics review and/or privacy assessment is needed. For details of our legal basis for using personal data and the rights you have over your data please see the University's privacy information. Please see the guidance on the ICS Ethics and Privacy website on what happens on submission.

Z0. Which is your main department?

- Information and Computing Science

Z1. Your full name:
Gertjan Brouwer

Z2. Your email address:
g.brouwer1@students.uu.nl

Z3. In what context will you conduct this research?

- As a student for my master thesis, supervised by:: Matthijs Vákár

Z5. Master programme for which you are doing the thesis

- Computing Science

Z6. Email of the course coordinator or supervisor (so that we can inform them that you filled this out and provide them with a summary):
m.i.l.vakar@uu.nl

Z7. Email of the moderator (as provided by the coordinator of your thesis project):
i.lykourentzou@uu.nl

Z8. Title of the research project/study for which you filled out this Quick Scan:
Variational inference in Stan

Z9. Summary of what you intend to investigate and how you will investigate this (200 words max):
Stan is a probabilistic programming language in which a probabilistic model can be specified. Inference in such models is usually done using a method called Markov chain Monte Carlo (MCMC) or a variant thereof. More recently a new branch of inference algorithms has been devised, namely Variational Inference(VI) methods. These algorithms deliver a result faster than their MCMC counterpart, at the cost of accuracy. However, the quality of the VI method is sometimes severely lacking. This is due to numerous factors such as: model misspecification, bad step size, bad variational family, etc. Recent research has tackled some of these problems. However, it is common that a solution presented in a paper will not work in the general case, but only on models provided by the paper. In this thesis some recent solutions to the problems of VI will be implemented in Stan. This implementation will be tested using multiple methods present in literature these days. This should provide us with a working version of VI in Stan which is properly tested against a benchmark of diverse models and data. As well as tested with multiple, newly, implemented diagnostics to provide feedback on model fit by the VI algorithm. These diagnostics will also guide future use of VI in Stan by providing meaningful measures of fit to a user.

Z10. In case you encountered warnings in the survey, does supervisor already have ethical approval for a research line that fully covers your project?

- Not applicable

Scoring

- Privacy: 0
- Ethics: 0

C Pathfinder experiment

This appendix has been added to compare VI to Pathfinder using MCMC as a golden standard. Note that no discussion or conclusion has been based on this data, and therefore, it has been added as an appendix.

dogs-dogs					
Mean			Std Dev		
VI	MCMC	Pathfinder	VI	MCMC	Pathfinder
1.800	1.809	1.787	0.104	0.230	0.234
−0.359	−0.359	−0.356	0.026	0.037	0.038
−0.210	−0.211	−0.209	0.018	0.043	0.044

Table 18

dogs-dogs_hierarchical					
Mean			Std Dev		
VI	MCMC	Pathfinder	VI	MCMC	Pathfinder
2.495	**0.923**	0.924	**0.129**	**0.011**	0.011
1.298	**0.786**	0.785	**0.094**	**0.019**	0.019

Table 19

garch-garch11					
Mean			Std Dev		
VI	MCMC	Pathfinder	VI	MCMC	Pathfinder
5.058	5.050	5.051	0.117	0.123	0.124
0.342	1.475	1.477	0.202	0.569	0.567
0.330	0.569	0.558	0.461	0.126	0.119
0.834	**0.291**	0.292	**0.563**	**0.123**	0.120

Table 20

arK-arK					
Mean			Std Dev		
VI	MCMC	Pathfinder	VI	MCMC	Pathfinder
−0.001	−0.001	−0.000	0.011	0.011	0.011
0.691	0.692	0.685	0.021	0.071	0.053
0.440	0.440	0.435	0.021	0.087	0.070
0.105	0.105	0.120	0.021	0.092	0.074
−0.036	−0.036	−0.029	0.021	0.086	0.069
−0.301	−0.301	−0.314	0.021	0.069	0.065
0.151	0.150	0.150	0.008	0.008	0.008

Table 21

diamonds-diamonds					
Mean			Std Dev		
VI	MCMC	Pathfinder	VI	MCMC	Pathfinder
6.645	6.660	6.591	0.004	0.251	0.057
6.244	6.363	6.422	0.012	0.328	0.122
−4.560	−4.684	−4.742	0.012	0.321	0.141
1.450	1.447	1.472	0.009	0.141	0.022
0.134	0.134	0.134	0.005	0.007	0.006
−0.040	−0.040	−0.040	0.004	0.007	0.005
0.023	0.023	0.023	0.004	0.006	0.004
0.002	0.002	0.002	0.004	0.004	0.003
−0.445	−0.445	−0.445	0.005	0.006	0.006
−0.093	−0.093	−0.093	0.005	0.006	0.006
−0.013	−0.013	−0.013	0.005	0.005	0.006
0.011	0.011	0.011	0.004	0.005	0.003
−0.001	−0.001	−0.001	0.004	0.005	0.003
0.001	0.001	0.001	0.004	0.004	0.003

Table 22

sblrc-blr					
Mean			Std Dev		
VI	MCMC	Pathfinder	VI	MCMC	Pathfinder
1.000	1.000	1.000	0.001	0.001	0.001
0.999	0.999	0.999	0.001	0.001	0.001
0.998	0.998	0.998	0.001	0.001	0.001
0.999	0.999	0.999	0.001	0.001	0.001
0.999	0.999	0.999	0.000	0.001	0.001
1.041	1.042	1.048	0.004	0.077	0.086

Table 23

sblri-blr					
Mean			Std Dev		
VI	MCMC	Pathfinder	VI	MCMC	Pathfinder
0.999	0.999	0.999	0.001	0.001	0.001
1.000	1.000	1.000	0.001	0.001	0.001
1.000	1.000	1.000	0.001	0.001	0.001
1.001	1.001	1.001	0.001	0.001	0.001
1.002	1.002	1.001	0.001	0.001	0.001
0.961	0.963	0.965	0.004	0.071	0.069

Table 24

low_dim_gauss_mix-low_dim_gauss_mix					
Mean			Std Dev		
VI	MCMC	Pathfinder	VI	MCMC	Pathfinder
−2.733	−2.734	−2.734	0.033	0.042	0.040
1.723	**2.870**	2.869	**0.009**	**0.055**	0.056
1.028	1.028	1.027	0.029	0.031	0.033
1.024	1.024	1.020	0.039	0.040	0.043
0.496	**0.622**	0.621	**0.066**	**0.015**	0.014

Table 25

eight_schools-eight_schools_centered					
Mean			Std Dev		
VI	MCMC	Pathfinder	VI	MCMC	Pathfinder
7.127	6.507	0.099	5.242	5.649	0.729
5.171	5.080	0.133	4.798	4.704	0.728
3.451	3.959	0.033	5.278	5.350	0.829
4.758	4.876	0.066	4.928	4.839	0.752
2.797	3.619	0.198	4.668	4.722	0.734
3.670	4.156	0.057	4.907	4.908	0.725
7.619	6.573	0.153	4.876	5.112	0.762
4.877	4.965	0.190	5.322	5.391	0.663
4.279	4.493	0.136	1.825	3.227	0.651
1.772	3.931	0.383	0.241	3.164	0.183

Table 26

GLM_Binomial					
Mean			Std Dev		
VI	MCMC	Pathfinder	VI	MCMC	Pathfinder
1.015	1.014	0.586	0.041	0.065	0.028
−0.173	−0.173	0.604	0.073	0.073	0.025
−0.921	−0.922	0.621	0.096	0.146	0.022

Table 27

M0_model					
Mean			Std Dev		
VI	MCMC	Pathfinder	VI	MCMC	Pathfinder
−0.311	**0.423**	0.423	**0.148**	**0.039**	0.040
−0.009	**0.497**	0.497	**0.137**	**0.038**	0.039

Table 28

wells_dist					
Mean			Std Dev		
VI	MCMC	Pathfinder	VI	MCMC	Pathfinder
0.610	0.607	0.607	0.037	0.061	0.062
−0.006	−0.006	−0.006	0.002	0.001	0.001

Table 29

gp_pois_regr-gp_regr					
Mean			Std Dev		
VI	MCMC	Pathfinder	VI	MCMC	Pathfinder
6.884	6.902	6.860	1.306	1.300	1.265
2.440	2.434	2.477	0.789	0.789	0.840
1.832	1.833	1.831	0.512	0.500	0.519

Table 30

dugongs					
Mean			Std Dev		
VI	MCMC	Pathfinder	VI	MCMC	Pathfinder
2.636	2.654	2.656	0.018	0.073	0.070
0.959	0.973	0.968	0.045	0.077	0.069
0.951	**0.863**	0.865	**0.092**	**0.033**	0.029
109.168	109.461	109.117	30.272	31.962	29.573

Table 31

hmm_example					
Mean			Std Dev		
VI	MCMC	Pathfinder	VI	MCMC	Pathfinder
0.727	0.666	0.671	0.472	0.100	0.099
−2.612	**0.334**	0.329	**0.430**	**0.100**	0.099
1.106	**0.073**	0.072	**0.032**	**0.029**	0.028
1.758	**0.927**	0.928	**0.019**	**0.029**	0.028

Table 32

surgical					
Mean			Std Dev		
VI	MCMC	Pathfinder	VI	MCMC	Pathfinder
−2.551	−2.553	−2.568	0.115	0.152	0.142
0.189	0.186	0.176	0.081	0.160	0.112
−2.945	−2.952	−3.037	0.341	0.435	0.375
−2.158	−2.184	−2.191	0.221	0.235	0.233
−2.610	−2.605	−2.581	0.265	0.273	0.261
−2.785	−2.774	−2.776	0.139	0.144	0.131
−2.968	−2.942	−2.944	0.247	0.280	0.263
−2.627	−2.622	−2.625	0.230	0.232	0.223
−2.675	−2.667	−2.713	0.259	0.263	0.265
−1.948	−1.981	−1.954	0.182	0.208	0.192
−2.618	−2.607	−2.599	0.227	0.228	0.202
−2.500	−2.496	−2.534	0.276	0.276	0.291
−2.167	−2.187	−2.183	0.184	0.192	0.180
−2.630	−2.624	−2.628	0.185	0.185	0.189

Table 33

References

[1] Matplotlib pyplot.boxplot. https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.boxplot.html. Accessed: 2024-07-10.

[2] Stan development repository. https://github.com/stan-dev/stan. Accessed: 2024-05-24.

[3] Abhinav Agrawal, Daniel R Sheldon, and Justin Domke. Advances in Black-Box VI: Normalizing Flows, Importance Weighting, and Optimization. Curran Associates, Inc., 2020.

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, 2006.

[5] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational Inference: A Review for Statisticians. *Journal of the American Statistical Association*, (518), April 2017.

[6] Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance Weighted Autoencoders, 2016.

[7] Bob Carpenter, Andrew Gelman, Matthew D. Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus A. Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. Stan: A Probabilistic Programming Language. *Journal of statistical software*, 2017.

[8] Bob Carpenter, Matthew D. Hoffman, Marcus Brubaker, Daniel Lee, Peter Li, and Michael Betancourt. The stan math library: Reverse-mode automatic differentiation in c++. 2015.

[9] George Casella and Roger L. Berger. *Statistical Inference*. Duxbury Advanced Series in Statistics and Decision Sciences. Thomson Learning, Australia, 2nd ed edition, 2002.

[10] Chris Chatfield. Model Uncertainty, Data Mining and Statistical Inference. *Journal of the Royal Statistical Society Series A: Statistics in Society*, (3), May 1995.

[11] Akash Kumar Dhaka, Alejandro Catalina, Michael R Andersen, Måns Magnusson, Jonathan Huggins, and Aki Vehtari. Robust, Accurate Stochastic Optimization for Variational Inference. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2020.

[12] Akash Kumar Dhaka, Alejandro Catalina, Manushi Welandawe, Michael R Andersen, Jonathan Huggins, and Aki Vehtari. Challenges and Opportunities in High Dimensional Variational Inference. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2021.

[13] Justin Domke and Daniel R Sheldon. Importance Weighting and Variational Inference. Curran Associates, Inc., 2018.

[14] Justin Domke and Daniel R Sheldon. Divide and Couple: Using Monte Carlo Variational Objectives for Posterior Approximation. Curran Associates, Inc., 2019.

[15] John Duchi, Elad Hazan, and Yoram Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization.

[16] Andrew Gelman, John B Carlin, Hal S Stern, and Donald B Rubin. *Bayesian data analysis*. Chapman and Hall/CRC, 1995.

[17] Andrew Gelman and Jennifer Hill. *Data analysis using regression and multilevel/hierarchical models*. Cambridge University Press, 2006.

[18] Andrew Gelman and Donald B. Rubin. Inference from Iterative Simulation Using Multiple Sequences. *Statistical Science*, (4), November 1992.

[19] Matthew D. Hoffman, David M. Blei, Chong Wang, and John Paisley. Stochastic Variational Inference. *Journal of Machine Learning Research*, (40), 2013.

[20] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, January 2017.

[21] T. Kloek and H. K. van Dijk. Bayesian Estimates of Equation System Parameters: An Application of Integration by Monte Carlo. *Econometrica*, (1), 1978.

[22] Alp Kucukelbir, Rajesh Ranganath, Andrew Gelman, and David Blei. Automatic Variational Inference in Stan. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2015.

[23] Alp Kucukelbir, Dustin Tran, Rajesh Ranganath, Andrew Gelman, and David M. Blei. Automatic Differentiation Variational Inference. *Journal of Machine Learning Research*, (14), 2017.

[24] S. Kullback and R. A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 1951.

[25] Maren Mahsereci and Philipp Hennig. Probabilistic Line Searches for Stochastic Optimization.

[26] Anton Mallasto, Augusto Gerolin, and Hà Quang Minh. Entropy-regularized 2-Wasserstein distance between Gaussian measures. *Information Geometry*, (1), July 2022.

[27] Nicholas Metropolis and S. Ulam. The Monte Carlo Method. *Journal of the American Statistical Association*, 1949.

[28] Thomas Minka. Divergence measures and message passing. *Technical report, Microsoft Research*, 2005.

[29] Shakir Mohamed, Mihaela Rosca, Michael Figurnov, and Andriy Mnih. Monte Carlo gradient estimation in machine learning. *The Journal of Machine Learning Research*, 2020.

[30] Emmanuel Okewu, Philip Adewole, and Oladipupo Sennaike. Experimental Comparison of Stochastic Optimizers in Deep Learning. In Sanjay Misra, Osvaldo Gervasi, Beniamino Murgante, Elena Stankova, Vladimir Korkhov, Carmelo Torre, Ana Maria A.C. Rocha, David Taniar, Bernady O. Apduhan, and Eufemia Tarantino, editors, *Computational Science and Its Applications – ICCSA 2019*, Cham, 2019. Springer International Publishing.

[31] Manfred Opper and David Saad. *Advanced Mean Field Methods: Theory and Practice*. MIT Press, 2001.

[32] Georg Ch. Pflug. Non-asymptotic confidence bounds for stochastic approximation algorithms with constant step size. *Monatshefte für Mathematik*, (3), September 1990.

[33] B. T. Polyak and A. B. Juditsky. Acceleration of Stochastic Approximation by Averaging. *SIAM Journal on Control and Optimization*, (4), July 1992.

[34] B. T. Polyak and A. B. Juditsky. Acceleration of Stochastic Approximation by Averaging. *SIAM Journal on Control and Optimization*, (4), July 1992.

[35] Rajesh Ranganath, Sean Gerrish, and David Blei. Black Box Variational Inference. PMLR, April 2014.

[36] Danilo Rezende and Shakir Mohamed. Variational Inference with Normalizing Flows. In *Proceedings of the 32nd International Conference on Machine Learning*. PMLR, June 2015.

[37] Herbert Robbins and Sutton Monro. A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 1951.

[38] Geoffrey Roeder, Yuhuai Wu, and David Duvenaud. Sticking the Landing: Simple, Lower-Variance Gradient Estimators for Variational Inference, 2017.

[39] Filippo Santambrogio. *Optimal Transport for Applied Mathematicians: Calculus of Variations, PDEs, and Modeling*. Progress in Nonlinear Differential Equations and Their Applications. Springer International Publishing, Cham, 2015.

[40] Stan-dev. Database with posteriors of interest for bayesian inference. https://github.com/stan-dev/posteriordb. Accessed: (24-01-24).

[41] George Tucker, Dieterich Lawson, Shixiang Gu, and Chris J. Maddison. Doubly Reparameterized Gradient Estimators for Monte Carlo Objectives, 2018.

[42] Aki Vehtari, Andrew Gelman, and Jonah Gabry. Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. *Statistics and Computing*, 2017.

[43] Aki Vehtari, Andrew Gelman, Daniel Simpson, Bob Carpenter, and Paul-Christian Bürkner. Rank-normalization, folding, and localization: An improved $\hat{R}$ for assessing convergence of MCMC. *Bayesian Analysis*, (2), June 2021.

[44] Aki Vehtari, Daniel Simpson, Andrew Gelman, Yuling Yao, and Jonah Gabry. Pareto Smoothed Importance Sampling, 2015.

[45] Manushi Welandawe, Michael Riis Andersen, Aki Vehtari, and Jonathan H. Huggins. Robust, Automated, and Accurate Black-box Variational Inference, March 2022.

[46] John Winn, Christopher M Bishop, and Tommi Jaakkola. Variational message passing. *Journal of Machine Learning Research*, 6(4), 2005.

[47] Yuling Yao, Aki Vehtari, Daniel Simpson, and Andrew Gelman. Yes, but Did It Work?: Evaluating Variational Inference. PMLR, 2018.

[48] Ersan Yazan and M. Fatih Talu. Comparison of the stochastic gradient descent based optimization techniques. In *2017 International Artificial Intelligence and Data Processing Symposium (IDAP)*, September 2017.

[49] Cheng Zhang, Judith Bütepage, Hedvig Kjellström, and Stephan Mandt. Advances in Variational Inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (8), August 2019.

[50] Lu Zhang, Bob Carpenter, Andrew Gelman, and Aki Vehtari. Pathfinder: Parallel quasi-Newton variational inference. *The Journal of Machine Learning Research*, (1), 2022.

[51] Pengchuan Zhang, Hunter Lang, Qiang Liu, and Lin Xiao. Statistical Adaptive Stochastic Gradient Methods, 2020.