PARTICLE-BASED VARIATIONAL INFERENCE WITH PRECONDITIONED FUNCTIONAL GRADIENT FLOW

Anonymous authors
Paper under double-blind review

ABSTRACT

Particle-based variational inference (VI) minimizes the KL divergence between model samples and the target posterior with gradient flow estimates. With the popularity of Stein variational gradient descent (SVGD), the focus of particle-based VI algorithms have been on the properties of functions in Reproducing Kernel Hilbert Space (RKHS) to approximate the gradient flow. However, the requirement of RKHS restricts the function class and algorithmic flexibility. This paper remedies the problem by proposing a general framework to obtain tractable functional gradient flow estimates. The functional gradient flow in our framework can be defined by a general functional regularization term that includes the RKHS norm as a special case. We also use our framework to propose a new particle-based VI algorithm: *preconditioned functional gradient flow* (PFG). Compared with SVGD, the proposed preconditioned functional gradient method has several advantages: larger function classes; greater scalability in the large particle-size scenarios; better adaptation to ill-conditioned target distribution; provable continuous-time convergence in KL divergence. Non-linear function classes such as neural networks can be incorporated to estimate the gradient flow. Both theoretical and experiments have shown the effectiveness of our framework.

1 INTRODUCTION

Sampling from unnormalized density is a fundamental problem in machine learning and statistics, especially for posterior sampling. Markov Chain Monte Carlo (MCMC) (Welling & Teh, 2011; Hoffman et al., 2014; Chen et al., 2014) and Variational inference (VI) (Ranganath et al., 2014; Jordan et al., 1999; Blei et al., 2017) are two mainstream solutions: MCMC is asymptotically unbiased but sample-exhausted; VI is computationally efficient but usually biased. Recently, particle-based VI algorithms (Liu & Wang, 2016; Detommaso et al., 2018; Liu et al., 2019) tend to minimize the Kullback-Leibler (KL) divergence between particle samples and the posterior, and absorb the advantages of both MCMC and VI: (1) non-parametric flexibility and asymptotic unbiasedness; (2) sample efficiency with the interaction between particles; (3) deterministic updates. Thus, these algorithms are competitive in sampling tasks, such as Bayesian inference (Liu & Wang, 2016; Feng et al., 2017; Detommaso et al., 2018), probabilistic models (Wang & Liu, 2016; Pu et al., 2017).

Given a target distribution $p_*(x)$, particle-based VI aims to find $g(t,x)$, so that starting with $X_0 \sim p_0$, the distribution $p(t,x)$ of the following method: $dX_t = g(t, X_t)dt$, converges to $p_*(x)$ as $t \to \infty$. By continuity equation (Jordan et al., 1998), we can capture the evolution of $p(t,x)$ by

$$\frac{\partial p(t,x)}{\partial t} = -\nabla \cdot (p(t,x)g(t,x)) \,. \tag{1}$$

In order to measure the "closeness" between $p(t,\cdot)$ and p_*, we typically adopt the KL divergence,

$$D_{\mathrm{KL}}(t) = \int p(t,x) \ln \frac{p(t,x)}{p_*(x)} dx. \tag{2}$$

Using chain rule and integration by parts, we have

$$\frac{dD_{\mathrm{KL}}(t)}{dt} = -\int p(t,x)[\nabla \cdot g(t,x) + g(t,x)^\top \nabla_x \ln p_*(x)]dx, \tag{3}$$

which captures the evolution of KL divergence.

To minimize the KL divergence, one needs to define a "gradient" to update the particle distribution as our $g(t, x)$. The most standard approach, *Wasserstein gradient* (Ambrosio et al., 2005), defines a gradient for $p(t, x)$ in the Wasserstein space, which contains probability measures with bounded second moment. In particular, for any functional $\mathcal{L}$ that maps probability density $p(t, x)$ to a non-negative scalar, we say that the particle density $p(t, x)$ follows the Wasserstein gradient flow of $\mathcal{L}$ if $g(t, x)$ is the $L^2(\mathbb{R}^d)$-functional derivative of $\mathcal{L}$. For KL divergence, the corresponding derivative is $\nabla \ln \frac{p_*(x)}{p(t,x)}$. However, the computation of deterministic and time-inhomogeneous Wasserstein gradient is non-trivial. It is necessary to restrict the function class of $g(t, x)$ to obtain a tractable form.

Stein variational gradient descent (SVGD) is the most popular particle-based algorithm, which provides a tractable form to update particles with the kernelized gradient flow (Chewi et al., 2020; Liu, 2017). It updates particles by minimizing the KL divergence with a functional gradient measured in RKHS. By restricting the functional gradient with bounded RKHS norm, it has an explicit formulation: $g(t, x)$ can be obtained by minimizing Eq. (3). Nonetheless, there are still some limitations due to the restriction of RKHS: (1) the expressive power is limited because kernel method is known to suffer from curse of dimensionality (Geenens, 2011); (2) with n particles, the $O(n^2)$ computational overhead of kernel matrix is required. Further, we identify another crucial limitation of SVGD: the kernel design is highly non-trivial. Even in the simple Gaussian case, where particles start with $\mathcal{N}(0, I)$ and $p_* = \mathcal{N}(\mu_*, \Sigma_*)$, commonly used kernels such as linear and RBF kernel, have fundamental drawbacks in SVGD algorithm (Example 1).

Our motivation originates from functional gradient boosting (Friedman, 2001; Nitanda & Suzuki, 2018; Johnson & Zhang, 2019). For each $p(t, x)$, we find a proper function as $g(t, x)$ in the function class $\mathcal{F}$ to minimize Eq. (3). In this context, we design a regularizer for the functional gradient to approximate variants of "gradient" explicitly. We propose a family of regularization to penalize the functional gradient output in the particle distribution. For well-conditioned $-\nabla^2 \ln p_*$[1], we can approximate the Wasserstein gradient directly; For ill-conditioned $-\nabla^2 \ln p_*$, we can adapt our regularizer to approximate a preconditioned one. Our functional gradient is an approximation to the preconditioned Wasserstein gradient. Regarding the function space, we do not restrict the function in RKHS. Instead, we use neural networks as our function classes to obtain better approximation capacity. The extension to the function space of neural networks gives the algorithm a much better expressive capacity, which will be justified with our empirical results.

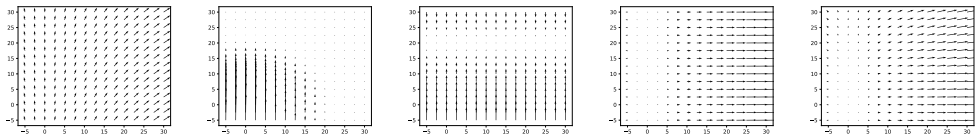
Contributions. We propose a particle-based VI framework with regularized functional gradient flow. Specifically, we choose a special family of regularizers to approximate preconditioned Wasserstein gradient, which is more effective than SVGD: The capacity of non-linear function classes (such as neural networks) are more expressive than RKHS; The functional gradient in our framework explicitly approximate the preconditioned Wasserstein gradient, which supports ill-conditioned cases and obtains provable convergence rate; Our proposed algorithm does not need the $O(n^2)$ kernel matrix, leading to computational efficiency when particle size is large. Both theoretical and empirical results show the effectiveness of our framework.

2 Analysis

Notations. In this paper, we use x to denote particle samples in $\mathbb{R}^d$. The distributions are assumed to be absolutely continuous w.r.t. Lebesgue measure. The probability density function of the posterior and is denoted by p_*. $p(t, x)$ (or p_t) refers to particle distribution at time t. For scalar function $p(t, x)$, $\nabla_x p(t, x)$ denotes its gradient w.r.t. x. For vector function $g(t, x)$, $\nabla_x g(t, x)$, $\nabla_x \cdot g(t, x)$, $\nabla_x^2 g(t, x)$ denote its Jacobian matrix, divergence, Hessian w.r.t. x, respectively. $\frac{\partial p(t,x)}{\partial t}$ and $\frac{\partial g(t,x)}{\partial t}$ denotes the partial derivative w.r.t. t. Without ambiguity, ∇ stands for ∇_x for conciseness. Notation $\|x\|_H^2$ stands for $x^\top H x$ and $\|x\|_I$ is denoted by $\|x\|$. Notation $\|\cdot\|_{\mathcal{H}^d}$ denotes the RKHS norm on $\mathbb{R}^d$.

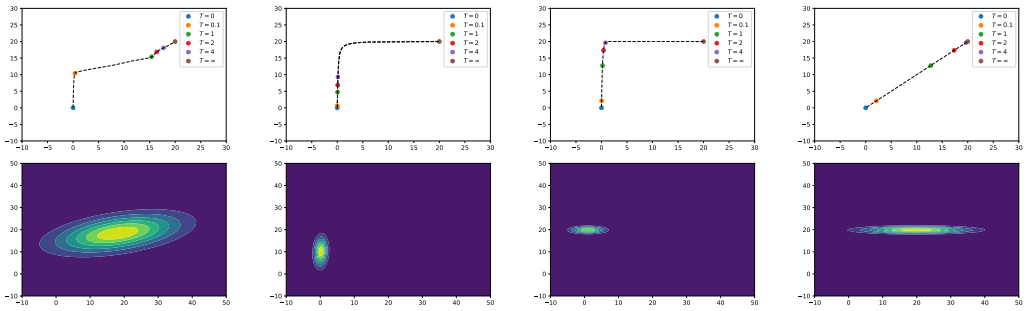
We let $g(t, x)$ belong to a vector-valued function class $\mathcal{F}$, and find the best functional gradient direction. Inspired by the gradient boosting algorithm for regression and classificaiton problems, we approximate the Wasserstein gradient flow at any time (t, x) by a function $g(t, x) \in \mathcal{F}$ which solves

[1] For any matrix, condition number is the ratio of the maximal to the minimal eigenvalues. A low condition number is said to be well-conditioned, while a high condition number is said to be ill-conditioned.



(a) $\frac{1}{2}\|f\|^2_{\mathcal{H}^d}$ (linear) (b) $\frac{1}{2}\|f\|^2_{\mathcal{H}^d}$ (RBF) (c) $\frac{1}{2}\mathbf{E}_{p_t}\|f\|^2$ (d) $\frac{1}{2}\mathbf{E}_{p_t}\|f\|^2_{\Sigma_*^{-1}}$ (e) Optimal transport

Figure 1: Illustration of the functional gradient $g(t,x)$ compared with optimal transport. (a)-(d) denotes the corresponding Q of the regularized functional gradient. (a)-(b) are SVGD algorithms with linear and RBF kernel. Optimal transport denotes the direction of the shortest path towards p_*.



(a) $Q(f) = \frac{1}{2}\|f\|^2_{\mathcal{H}^d}$ (linear) (b) $Q(f) = \frac{1}{2}\|f\|^2_{\mathcal{H}^d}$ (RBF) (c) $Q(f) = \frac{1}{2}\mathbf{E}_{p_t}\|f\|^2$ (d) $Q(f) = \frac{1}{2}\mathbf{E}_{p_t}\|f\|^2_{\Sigma_*^{-1}}$

Figure 2: Evolution of particle distribution from $\mathcal{N}([0,0]^\top, I)$ to $\mathcal{N}([20,20]^\top, \mathrm{diag}(100,1))$ (first row: evolution of particle mean μ_t; second row: particle distribution $p(5,x)$ at $t=5$)

the following minimization formulation:

$$g(t,x) = \arg\min_{f\in\mathcal{F}} \left[-\int p(t,x)[\nabla\cdot f(x) + f(x)^\top \nabla \ln p_*(x)]dx + Q(f) \right], \tag{4}$$

where $Q(\cdot)$ is a regularization term that limit the output magnitude of f. This regularization term also implicitly determines the underlying "distance metric" used to define the gradient estimates $g(t,x)$ in our framework. When $Q(x) = \frac{1}{2}\int p(t,x)\|f(x)\|^2 dx$, Eq. (4) is equivalent to

$$g(t,x) = \arg\min_{f\in\mathcal{F}} \int p(t,x) \left\| f(x) - \nabla \ln \frac{p_*(x)}{p_t(x)} \right\|^2 dx. \tag{5}$$

If $\mathcal{F}$ is well-specified, i.e., $\nabla \ln \frac{p_*(x)}{p_t(x)} \in \mathcal{F}$, we have $g(t,x) = \nabla \ln \frac{p_*(x)}{p_t(x)}$, which is the direction of Wasserstein gradient. Interestingly, despite the computational intractability of Wasserstein gradient, Eq. (4) provides a tractable variational approximation.

For RKHS, we will show that SVGD is a special case of Eq. (4), where $Q(f) = \frac{1}{2}\|f\|^2_{\mathcal{H}^d}$ (Sec. 3.2.1). We point out that RKHS norm cannot regularize the functional gradient properly since it is fixed for any $p(t,x)$. Our next example shows that SVGD is a suboptimal solution for Gaussian case.

Example 1. *Consider that $p(t,\cdot)$ is $\mathcal{N}(\mu_t, \Sigma_t)$, p_* is $\mathcal{N}(\mu_*, \Sigma_*)$. We consider the SVGD algorithm with linear kernel, RBF kernel, and regularized functional gradient formulation with $Q(f) = \frac{1}{2}\mathbf{E}_{p_t}\|f\|^2$, and $Q(f) = \frac{1}{2}\mathbf{E}_{p_t}\|f\|^2_{\Sigma_*^{-1}}$. Starting with $\mathcal{N}(0,I)$, Fig. 1 plots the $g(0,x)$ with different Q and the optimal transport direction; the path of μ_t and $p(5,x)$ are illustrated in Fig. 2. The detailed mathematical derivation are provided in the Appendix.*

Example 1 shows the comparison of different regularizations for the functional gradient. For RKHS norm, we consider the most commonly used kernels: linear and RBF. Fig. 1 shows the $g(0,x)$ of different regularizers: it is clear that only $Q(f) = \frac{1}{2}\mathbf{E}_{p_t}\|f\|^2_{\Sigma_*^{-1}}$ approximates the optimal transport direction, while other $g(0,x)$ deviates significantly. SVGD with linear kernel contains curl component ($\mu_*\mu_t^\top$ is non-symmetric; See Appendix for details) to make p_t rotate. SVGD with RBF kernel even suffers from gradient vanishing in low-density area. Fig. 2 demonstrates the path of μ_t with different

regularizers. For linear kernel, due to the curl component, $p(5, x)$ is rotated with an angle (Fig. 2 (a)). For RBF kernel, it is misspecified, leading to slow convergence, since linear function is not contained in the function class. The L_2 regularizer shows suboptimal performance due to the ill-conditioned Σ_*. We can see that $Q(f) = \frac{1}{2}\mathbf{E}_{p_t}\|f\|^2_{\Sigma_*^{-1}}$ produces the optimal path for μ_t (the line between μ_0 and μ_*).

2.1 General Regularization

Inspired by the Gaussian case, we consider the general form

$$Q(f(x)) = \frac{1}{2}\int p(t,x)\|f(x)\|_H^2 dx \tag{6}$$

where H is a symmetric positive definite matrix. Proposition 1 shows that with Eq. (6), the resulting functional gradient is the approximation of preconditioned Wasserstein gradient. It also implies the existence of $g(t, x)$ when $\mathcal{F}$ is closed, since $\|\cdot\|_H$ is non-negative.

Proposition 1. *Consider the case that $Q(f) = \frac{1}{2}\int p(t,x)\|f(x)\|_H^2 dx$, where H is a symmetric positive definite matrix. Then the functional gradient defined by Eq.* (4) *is equivalent as*

$$g(t,x) = \arg\min_{f\in\mathcal{F}} \frac{1}{2}\int p(t,x)\left\|f(x) - H^{-1}\nabla \ln \frac{p_*(x)}{p(t,x)}\right\|_H^2 dx \tag{7}$$

Remark. Our regularizer is similar to the Bregman divergence in proximal gradient. We can further extend $Q(\cdot)$ with a convex function $h(\cdot) : \mathbb{R} \to [0, \infty)$, where the regularizer is defined by $Q(f(x)) = \mathbf{E}_{p_t} h(f(x))$. We can adapt more complex geometry in our framework with proper $h(\cdot)$.

2.2 Convergence analysis

Equilibrium condition. To provide the theoretical guarantees, we show that the stationary distribution of our $g(t, x)$ update is p_*. Meanwhile, the evolution of KL-divergence is well-defined (without the explosion of functional gradient) and descending. We list the regularity conditions below.

$[\mathbf{A_1}]$ (Regular function class) The tail of p_* and f is regular: $\lim_{\|x\|\to\infty} f(\theta, x)p_*(x) = 0$ for $f \in \mathcal{F}$. The discrepancy induced by function class $\mathcal{F}$ is positive-definite: for any $q \neq p$, there exists $f \in \mathcal{F}$ such that $\int q(x)[\nabla \cdot f(x) + f(x)^\top \nabla \ln p(x)]dx > 0$. For $f \in \mathcal{F}$ and $c \in \mathbb{R}$, $cf \in \mathcal{F}$.

$[\mathbf{A_2}]$ (L-Smoothness) For any $x \in \mathbb{R}^d$, $p_*(x) > 0$ and $p(t, x) > 0$ are twice differentiable L-smooth densities, such that $\|\nabla \ln p(x) - \nabla \ln p(y)\| \leq L\|x - y\|$, with $\mathbf{E}_{p_t}\|x\|^2 < \infty$.

Note that $[\mathbf{A_1}]$ is similar to the positive definite kernel in RKHS, which guarantees the decrease of KL divergence. $[\mathbf{A_2}]$ is designed to make the functional gradient well-defined: RHS of Eq. 3 is finite.

Proposition 2. *Under* $[\mathbf{A_1}]$, $[\mathbf{A_2}]$, *when we update X_t as Eq. (7), we have $-\infty < \frac{dD_{\mathrm{KL}}}{dt} < 0$ for all $p(t, x) \neq p_*(x)$.* i.e., *$g(t, x) = 0$ if and only if $p(t, x) = p_*(x)$.*

Proposition 2 shows that the continuous dynamics of our $g(t, x)$ is well-defined and the KL divergence along the dynamics is descending. The only stationary distribution for $p(t, x)$ is $p_*(x)$.

Convergence rate. The convergence rate of our framework mainly depends on (1) the capacity of function class $\mathcal{F}$; (2) the complexity of p_*. In this section, we analyze that when the approximation error is small and the target p_* is log-Sobolev, the KL divergence converges linearly.

$[\mathbf{A_3}]$ (ϵ-approximation) For any $t > 0$, there exists $f_t(x) \in \mathcal{F}$ and $\epsilon < 1$, such that

$$\int p(t,x)\left\|f_t(x) - H^{-1}\nabla \ln \frac{p(t,x)}{p_*(x)}\right\|_H^2 dx \leq \epsilon \int p(t,x)\left\|\nabla \ln \frac{p(t,x)}{p_*(x)}\right\|_{H^{-1}}^2 dx$$

$[\mathbf{A_4}]$ The target p_* satisfies μ-log-Sobolev inequality ($\mu > 0$): For any differentiable function g, we have $\mathbb{E}_{p_*}\left[g^2 \ln g^2\right] - \mathbb{E}_{p_*}\left[g^2\right] \ln \mathbb{E}_{p_*}\left[g^2\right] \leq \frac{2}{\mu}\mathbb{E}_{p_*}\left[\|\nabla g\|^2\right]$.

Specifically, $[\mathbf{A_3}]$ is the error control of gradient approximation. With Universal approximation theorem (Hornik et al., 1989), any continuous function can be estimated by neural networks, which indicates the existence of such a function class. $[\mathbf{A_4}]$ is a common assumption in sampling literature. It is more general than strongly concave assumption Vempala & Wibisono (2019).

Theorem 1. *Under* $[\mathbf{A_1}]$-$[\mathbf{A_4}]$*, for any* $t > 0$*, we assume that the largest eigenvalue,* $\lambda_{\max}(H) = m$*. Then we have* $\frac{dD_{\mathrm{KL}}(t)}{dt} \leq -\frac{1}{2}\mathbb{E}_{p_t}\left\|\nabla \ln \frac{p_t(x)}{p_*(x)}\right\|_{H^{-1}}^2$ *and* $D_{\mathrm{KL}}(t) \leq \exp(-(1-\epsilon)\mu t/m)D_{\mathrm{KL}}(0)$.

Theorem 1 shows that when p_* is log-Sobolev, our algorithm achieves linear convergence rate with a function class powerful enough to approximate the preconditioned Wasserstein gradient. Moreover, considering the discrete algorithm, we have to make sure that the step size is proper $\|H^{-1}\nabla^2 \ln p_*(x)\| \leq c$, for some constant $c > 0$. Thus, the preconditioned $H = -c\nabla^2 \ln p_*$ is better than plain one $H = cLI$, since $(-\nabla^2 \ln p_*)^{-1} \succeq L^{-1}I$. For better understanding, we have provided a Gaussian case discretized proof in Appendix A.6 to illustrate this phenomenon.

3 PFG: Preconditioned Functional Gradient Flow

Algorithm 1 PFG: Preconditioned Functional Gradient Flow

Input: Unnormalized target distribution $p_*(x) = e^{-U(x)}$, $f_\theta(x) : \mathbb{R}^d \to \mathbb{R}^d$, initial particles (parameters) $\{x_0^i\}_{i=1}^n$, θ_0, iteration parameter T, T', step size η, η', regularization function $h(\cdot)$.

for $t = 1, \ldots, T$ **do**
  Assign $\theta_t^0 = \theta_{t-1}$;
  for $t' = 1, \cdots, T'$ **do**
    Compute $\hat{L}(\theta) = \frac{1}{n}\sum_{i=1}^n \left(h(f_\theta(x_t^i)) + f_\theta(x_t^i)\cdot\nabla U(x_t^i) - \nabla\cdot f_\theta(x_t^i)\right)$
    Update $\theta_t^{t'} = \theta_t^{t'-1} - \eta'\nabla\hat{L}(\theta_t^{t'-1})$;
  end
  Assign $\theta_t = \theta_t^{T_1}$ and update particles $x_t^i = x_t^i + \eta\left(f_{\theta_t}(x_t^i)\right)$ for all $i = 1, \cdots, n$;
end

Return: Optimized particles $\{x_T^i\}_{i=1}^n$

3.1 Algorithm

We will realize our algorithm with parametric f_θ (such as neural networks) and discretize the update.

Parametric Function Class. We can let $\mathcal{F} = \{f_\theta(x)\}$ and apply $g(t,x) = f_{\hat{\theta}_t}(x)$, such that

$$\hat{\theta}_t = \arg\min_\theta \left[\int p(t,x)[-\nabla\cdot f_\theta(x) - f_\theta(x)^\top \nabla \ln p_*(x) + \frac{1}{2}\|f_\theta(x)\|_H^2]dx\right], \tag{8}$$

where H is a symmetric positive definite matrix. Note that Eq. (8) is a direct result from Eq. (4) and (6). The parametric function class allows f_θ to be optimized by iterative algorithms.

Choice of H. Considering the trajectory of posterior mean, it is equivalent to the conventional optimization, so that $-\nabla^2 \ln p_*$ is ideal (Newton's method) to implement discrete algorithms. We follow this idea and use diagonal estimation for efficient computation as Adagrad (Duchi et al., 2011).

Discrete update. We can present our algorithm by discretizing Eq. (4) and (8). Given $X_0 \sim p_0$, we update X_k as $X_{k+1} = X_k + \eta f_{\hat{\theta}_k}(X_k)$, where $\hat{\theta}_k$ is obtained by Eq. (8) with (stochastic) gradient descent. The integral over $p(k,x)$ is estimated by particle samples. Full procedure is presented in Alg. 1, where the regularizer h is $\frac{1}{2}\|\cdot\|_H^2$ by default.

3.2 Comparison with SVGD

3.2.1 SVGD from Functional Gradient View

For simplicity, we prove the case under finite-dimensional feature map[2], $\psi(x) : \mathbb{R}^d \to \mathbb{R}^h$. We assume that $\mathcal{F} = \{W\psi(x) : W \in \mathbb{R}^{d\times h}\}$, and let kernel $k(x,y) = \psi(x)^\top\psi(y)$, $Q(f) = \frac{1}{2}\|f\|_{\mathcal{H}^d}^2 = \frac{1}{2}\|W\|_F^2$, where RKHS norm is the Frobenius norm of W. The solution is defined by

$$\hat{W}_t = \arg\min_{W\in\mathbb{R}^{d\times h}} -\int p(t,x)\,\mathrm{trace}[W\nabla\psi(x) + W\psi(x)\nabla\ln p_*(x)^\top]dx + \frac{1}{2}\|W\|_F^2, \tag{9}$$

[2]Infinite-dimensional version is provided in the Appendix.

which gives $\hat{W}_t = \int p(t, x')[\nabla\psi(x')^\top + \nabla \ln p_*(x')\psi(x')^\top]dx'$. This implies that

$$g(t, x) = \hat{W}_t\psi(x) = \int p(t, y)[\nabla_y k(y, x) + \nabla \ln p_*(y)k(y, x)]dy, \tag{10}$$

which is equivalent to SVGD. For linear function classes, such as RKHS, Q can be directly applied to the parameters, such as W here. The regularization of SVGD is $\frac{1}{2}\|\cdot\|_F^2$ (the norm defined in RKHS). For non-linear function classes, such as neural networks, the RKHS norm cannot be defined.

3.2.2 Limitations of SVGD

Kernel function class. As in Section 3.2.1, RKHS only contains linear combination of the feature map functions, which suffers from curse of dimensionality (Geenens, 2011). On the other hand, some non-linear function classes, such as neural network, performs well on high dimensional data LeCun et al. (2015). The extension to non-linear function classes is needed for better performance.

Gradient preconditioning in SVGD. In Example 1, when $-\nabla^2 \ln p_*$ is ill-conditioned, PFG algorithm follows the shortest path from μ_0 to μ_*. Although SVGD can implement preconditioning matrices as (Wang et al., 2019), due to the curl component and time-dependent Jacobian of $d\mu_t/dt$, any symmetric matrix cannot provide optimal preconditioning (detailed derivation in Appendix).

Suboptimal convergence rate. For log-Sobolev p_*, SVGD with commonly used bounded smoothing kernels (such as RBF kernel) cannot reach the linear convergence rate (Duncan et al., 2019) and the explicit KL convergence rate is unknown yet. Meanwhile, the Wasserstein gradient flow converges linearly. When the function class is sufficiently large, PFG converges provably faster than SVGD.

Computational cost. For SVGD, main computation cost comes from the kernel matrix: with n particles, we need $O(n^2)$ memory and computation. Our algorithm uses an iterative approximation to optimize $g(t, x)$, whose memory cost is independent of n and computational cost is $O(n)$ (Bertsekas et al., 2011). The repulsive force between particles is achieved by $\nabla \cdot f$ operator on each particle.

4 Experiment

To validate the effectiveness of our algorithm, we have conducted experiments on both synthetic and real datasets. Without special declaration, we use parametric two-layer neural networks with Sigmoid activation as our function class. To approximate H in real datasets, we use the approximated diagonal Hessian matrix $\hat{H}$, and choose $H = \hat{H}^\alpha$, where $\alpha \in \{0, 0.1, 0.2, 0.5, 1\}$; the inner loop T' of PFG is chosen from $\{1, 2, 5, 10\}$, the hidden layer size is chosen from $\{32, 64, 128, 256, 512\}$. The parameters are chosen by validation. More detailed settings are provided in the Appendix.

Gaussian Mixture. To demonstrate the capacity of non-linear function class, we have conducted the Gaussian mixture experiments to show the advantage over linear function class, e.g., SVGD (RBF kernel). We consider to sample from a 10-cluster Gaussian Mixture distribution. Both SVGD and our algorithm are trained with 1,000 particles. Fig. 3 shows that the estimated score by RBF kernel is usually unsatisfactory: (1) In low-density area, it suffers from gradient vanishing, which makes samples stuck at these parts (similar to Fig. 1 (b)); (2) The score function cannot distinguish connected clusters. Specifically, some clusters are isolated while others might be connected. The choice of bandwidth is hard. The fixed bandwidth makes the SVGD algorithm un-

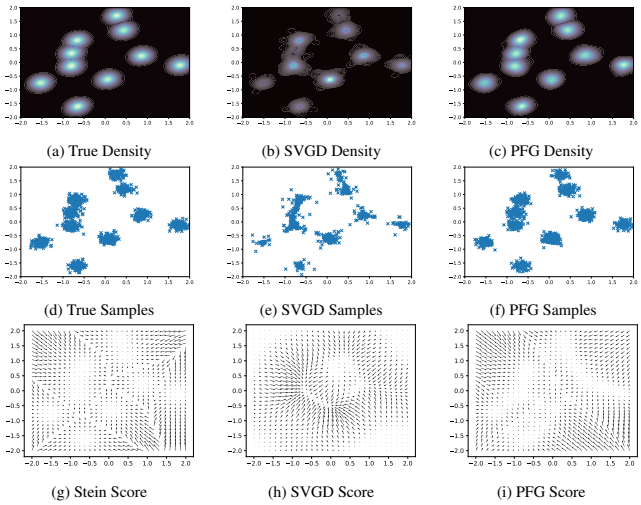
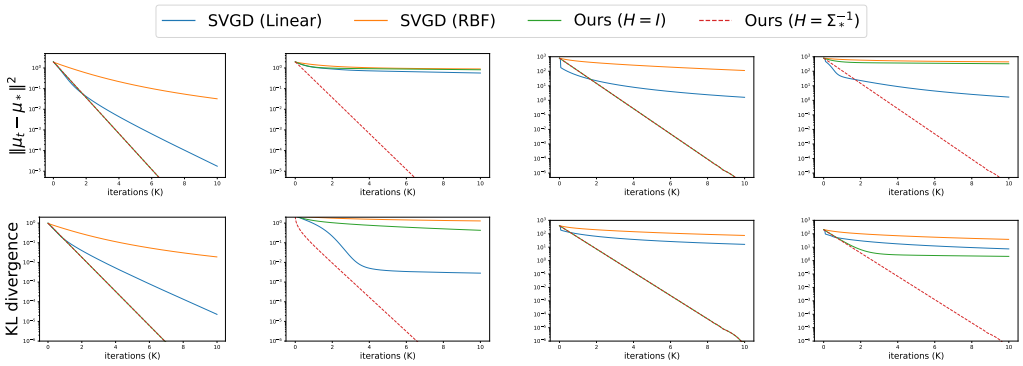


Figure 3: Particle-based VI for Gaussian mixture.



(a) $\mu_* = [1, 1]^\top$ $\Sigma_* = \mathrm{diag}(1, 1)$ (b) $\mu_* = [1, 1]^\top$ $\Sigma_* = \mathrm{diag}(100, 1)$ (c) $\mu_* = [20, 20]^\top$ $\Sigma_* = \mathrm{diag}(1, 1)$ (d) $\mu_* = [20, 20]^\top$ $\Sigma_* = \mathrm{diag}(100, 1)$

Figure 4: Evolution of particle distribution from $\mathcal{N}(0, I)$ to $\mathcal{N}(\mu_*, \Sigma_*)$ (first row: mean squared error of μ_t: $\|\mu_t - \mu_*\|^2$; second row: KL divergence between $p(t, x)$ and $p_*(x)$)

able to determine the gradient near connected clusters. It fails to capture the clusters near $(-0.75, 0.5)$. However, the non-linear function class is able to resolve the above difficulties. In PFG, we found that the score function is well-estimated and the resulting samples mimic the true density properly.

Ill-conditioned Gaussian distribution. We show the effectiveness of our proposed regularizer. For ill-conditioned Gaussian distribution, the condition number of Σ_* is large, *i.e.*, the ratio between maximal and minimal eigenvalue is large. We follow Example 1 and compare different μ_* and Σ_*. When Σ_* is well-conditioned ($\Sigma_* = I$), L_2 regularizer (equivalent to Wasserstein gradient) performs well. However, it will be slowed down significantly with ill-conditioned Σ_*. For SVGD with linear kernel, the convergence slows down with shifted μ_* or ill-conditioned Σ_*. For SVGD with RBF kernel, the convergence is slow due to the misspecified function class. Interestingly, for ill-conditioned case, μ_t of SVGD (linear) converges faster than our method with $H = I$ but KL divergence does not always follow the trend. The reason is that Σ_t of SVGD is highly biased, making KL divergence large. Our algorithm ($H = \Sigma_*^{-1}$) extends the Wasserstein gradient and makes the particle-based sampling algorithm compatible with ill-conditioned sampling case.

Logistic Regression. We conduct Bayesian logistic regression for binary classification on Sonar and Australian dataset (Dua & Graff, 2017). In particular, the prior of weight samples are assigned $\mathcal{N}(0, 1)$; the step-size is fixed as 0.01. We compared our algorithm with SVGD (200 particles), full gradient is used in this part. To measure the posterior approximation quality, we use 3 metrics: (1) the distance between sample mean μ_t and posterior mean μ_*; (2) the Maximum Mean Discrepancy (MMD) distance Gretton et al. (2012) from particle samples to posterior samples; (3) the evidence lower bound (ELBO) of current particles, $\mathbf{E}_{x \sim p_t}[\log p_*(x) - \log p(t, x)]$, where $p_*(x)$ is the unnormalized density $p(x, \mathcal{D})$ with training data $\mathcal{D}$, the entropy term $\log p(t, x)$ is estimated by kernel density. The ground truth of p_* is estimated by NUTS (Hoffman et al., 2014). Fig. 5 shows that our method outperforms SVGD consistently.

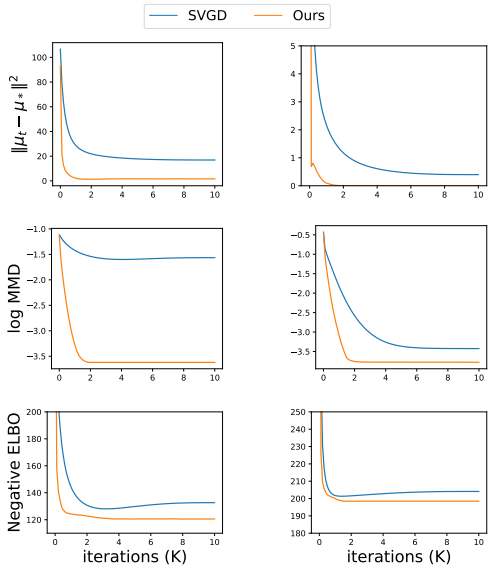


Figure 5: Posterior sampling for Bayesian logistic regression. (200 particles; dataset: *sonar* (first column) and *Australian* (second column). μ_t: particle mean; μ_*: posterior mean.)

Hierarchical Logistic Regression. For hierarchical logistic regression, we consider a 2-level hierarchical regression, where prior of weight samples is $\mathcal{N}(0, \alpha^{-1})$. The prior of α is

Table 3: Time Comparison on Bayesian logistic regression (sonar dataset, per 1,000 iteration)

# OF PARTICLES	5	10	100	1000	2000
SVGD	**5.20**$_{\pm 0.10}$	**5.25**$_{\pm 0.13}$	**5.71**$_{\pm 0.11}$	19.10$_{\pm 0.52}$	81.50$_{\pm 1.86}$
PFG	7.87$_{\pm 0.16}$	7.95$_{\pm 0.15}$	8.15$_{\pm 0.17}$	**11.56**$_{\pm 0.24}$	**13.22**$_{\pm 0.26}$

Gamma$(0, 0.01)$. All step-sizes are chosen from $\{10^{-t} : t = 1, 2, \cdots\}$ by validation. We compare the test accuracy and negative log-likelihood as (Liu & Wang, 2016), mini-batch was used to estimate the log-likelihood of training samples (batch-size = 200). Tab. 1 shows that our algorithm generally performs the best on this task.

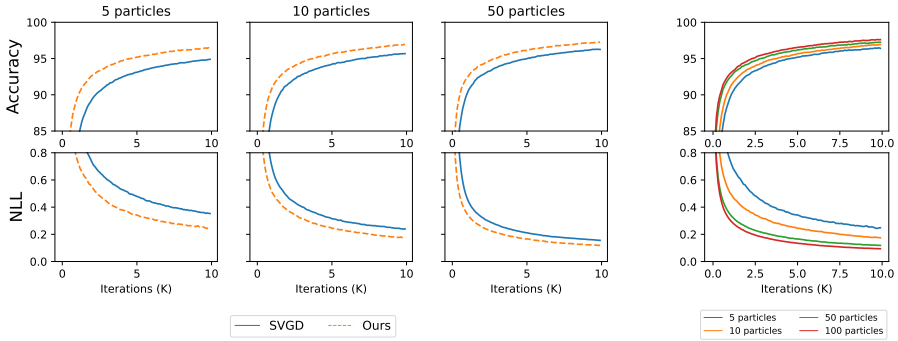
Bayesian Neural Networks. We compare our algorithm with SGLD (Welling & Teh, 2011) and SVGD (Liu & Wang, 2016) on Bayesian neural networks. Our experiment setting follows SVGD (Liu & Wang, 2016). In particular, we conduct the two-layer network with 50 hidden units (100 for Year dataset, 128 for MNIST) and ReLU activation function; the batch-size is 100 (1000 for Year dataset), All step-sizes are chosen from $\{10^{-t} : t = 1, 2, \cdots\}$ by validation. For UCI datasets, data samples are randomly partitioned to two parts: 90% (training), 10% (testing) and we use 100 particles for inference. For MNIST, data split follows the default setting, we compare $5, 10, 50, 100$ particles (SVGD with 100 particles exceeded the memory limit). In Tab. 2, PFG outperforms SVGD and SGLD significantly. In Fig. 6, we found that the accuracy and NLL of PFG is better than SVGD with all particle size. With more particle samples, PFG algorithm also improves.

Table 1: Hierarchical Logistic Regression on UCI datasets (particle size = 200)

	COVTYPE		GERMAN	
	ACC.	NLL	ACC.	NLL
SVGD	75.1	0.58	65.1	0.71
SGLD	74.9	0.57	64.8	0.65
PFG	**75.7**	**0.51**	**67.6**	**0.64**

Table 2: Averaged test root-mean-square error (RMSE) and test log-likelihood of Bayesian Neural Networks on UCI datasets (100 particles). Results are computed by 10 trials.

	AVERAGE TEST RMSE			AVERAGE TEST LOG-LIKELIHOOD		
DATASET	SVGD	SGLD	PFG	SVGD	SGLD	PFG
BOSTON	3.04$_{\pm 0.12}$	2.79$_{\pm 0.16}$	**2.47**$_{\pm 0.11}$	-2.76$_{\pm 0.15}$	-2.63$_{\pm 0.14}$	**-2.35**$_{\pm 0.12}$
CONCRETE	5.51$_{\pm 0.17}$	4.97$_{\pm 0.12}$	**4.74**$_{\pm 0.14}$	-3.07$_{\pm 0.23}$	-3.03$_{\pm 0.13}$	**-2.83**$_{\pm 0.16}$
ENERGY	1.96$_{\pm 0.10}$	0.84$_{\pm 0.07}$	**0.48**$_{\pm 0.04}$	-2.15$_{\pm 0.11}$	-1.82$_{\pm 0.16}$	**-1.28**$_{\pm 0.06}$
PROTEIN	4.89$_{\pm 0.11}$	4.61$_{\pm 0.13}$	**4.51**$_{\pm 0.06}$	-3.12$_{\pm 0.14}$	-2.99$_{\pm 0.14}$	**-2.89**$_{\pm 0.07}$
WINERED	0.68$_{\pm 0.03}$	0.67$_{\pm 0.05}$	**0.60**$_{\pm 0.02}$	-1.92$_{\pm 0.05}$	-1.81$_{\pm 0.08}$	**-1.61**$_{\pm 0.03}$
WINEWHITE	0.69$_{\pm 0.04}$	0.71$_{\pm 0.06}$	**0.59**$_{\pm 0.02}$	-1.85$_{\pm 0.04}$	-1.69$_{\pm 0.07}$	**-1.58**$_{\pm 0.04}$
YEAR	8.65$_{\pm 0.08}$	8.69$_{\pm 0.06}$	**8.56**$_{\pm 0.04}$	-3.54$_{\pm 0.10}$	-3.64$_{\pm 0.09}$	**-3.51**$_{\pm 0.03}$



(a) SVGD vs PFG on MNIST classification

(b) PFG with different particles

Figure 6: Test accuracy and NLL of Bayesian Neural Network (MNIST classification)

Time Comparison. As indicated, our algorithm is more scalable than SVGD in terms of particle numbers. Tab. 3 shows that the SVGD algorithm entails much more computation cost with the increase of particle numbers. Our functional gradient is obtained with iterative approximation. When n is large, our $O(n)$ algorithm is much faster than $O(n^2)$ kernel method.

5 Related Works

Stein Discrepancy and SVGD. Stein discrepancy (Liu et al., 2016; Chwialkowski et al., 2016; Liu & Wang, 2016) is known as the foundation of SVGD algorithm, which is defined on the bounded function classes, such as functions with bounded RKHS norm. We provide another view to derive SVGD: SVGD is a functional gradient method with RKHS norm regularizer. We suggest that the approximated Wasserstein gradient has many advantages over SVGD as the function class is more general and the kernel computation can be saved. We use neural networks as our $\mathcal{F}$ in real practice, which is much more efficient than SVGD. Moreover, the general regularizer in Eq. (6) improves the conditioning of functional gradient, which is proven better than SVGD in both theory and experiments. The preconditioning of our algorithm is more explainable than SVGD, without incorporating RKHS-dependent properties.

Learning Discrepancy with Neural Networks. Some recent works Hu et al. (2018); Grathwohl et al. (2020) leverage neural networks to learn the Stein discrepancy, which are related to our PFG algorithm. Grathwohl et al. (2020) implements $Q(f(x)) = \mathbf{E}_{p_t}\|f(x)\|^2$ (L_2 norm) in our framework to measure the similarity between two distributions and use the metric to train energy-based models (EBM) without likelihood. The proposed EBM training strategy is efficient and accurate thanks to the capacity of neural networks. However, they did not further validate the sampling performance of their "learned discrepancy", so it is a new version of KSD Liu et al. (2016), rather than SVGD. Hu et al. (2018) is also based on L_2 norm regularization to train a neural "generator" for sampling tasks. They discussed the scaling of L_2 regularization to align with the step size. In contrast, we restrict our investigation in particle-based VI and avoid learning a "generator", which may introduce more parameterization errors. Moreover, we extend the design of Q to a general case, and emphasize the benefits of our proposed regularizers corresponding to the preconditioned algorithm. We include the Example 1 and Figure 4 to demonstrate the necessity of preconditioning. We provide a Gaussian analysis of PFG algorithm in Appendix A.6 has demonstrated the improvement in discretized algorithm theoretically.

KL Wasserstein Gradient Flow Wasserstein gradient flow (Ambrosio et al., 2005) is the continuous dynamics to minimize functionals in Wasserstein space, which is crucial in sampling and optimal transport tasks. However, the numerical computation of Wasserstein gradient is non-trivial. Previous works (Peyré, 2015; Benamou et al., 2016; Carlier et al., 2017) attempt to find tractable formulation with spatial discretization, which suffers from curse of dimensionality. More recently, (Mokrov et al., 2021; Alvarez-Melis et al., 2021) leverage neural networks to model the Wasserstein gradient and aim to find the full transport path between p_0 and p_*. The computation of full path is extremely large. Salim et al. (2020) defines proximal gradient in Wasserstein space by JKO operator. However, the work mainly focus on theoretical properties and the efficient implementation remains open. When the functional is KL divergence, Wasserstein gradient can also be realized with Langevin dynamics (Welling & Teh, 2011; Bernton, 2018) by adding Gaussian noise to particles, which are also variants of MCMC. However, the deterministic algorithm to approximate Wasserstein gradient flow is still challenging. Our framework provides a tractable approximation to fix this gap.

6 Conclusion

In particle-based VI, we consider that the particles can be updated with a preconditioned functional gradient flow. Our theoretical results and Gaussian example led to an algorithm: PFG that can approximate preconditioned Wasserstein gradient directly. Our theory indicates that when the function class is large, PFG is provably faster than conventional particle-based VI, SVGD. The empirical result showed that PFG performs better than conventional SVGD and SGLD variants.

REFERENCES

David Alvarez-Melis, Yair Schiff, and Youssef Mroueh. Optimizing functionals on the space of probabilities with input convex neural networks. *arXiv preprint arXiv:2106.00774*, 2021.

Luigi Ambrosio, Nicola Gigli, and Giuseppe Savaré. *Gradient flows: in metric spaces and in the space of probability measures*. Springer Science & Business Media, 2005.

Jimmy Ba, Murat A Erdogdu, Marzyeh Ghassemi, Shengyang Sun, Taiji Suzuki, Denny Wu, and Tianzong Zhang. Understanding the variance collapse of svgd in high dimensions. In *International Conference on Learning Representations*, 2021.

Jean-David Benamou, Guillaume Carlier, Quentin Mérigot, and Edouard Oudet. Discretization of functionals involving the monge–ampère operator. *Numerische mathematik*, 134(3):611–636, 2016.

Espen Bernton. Langevin monte carlo and jko splitting. In *Conference On Learning Theory*, pp. 1777–1798. PMLR, 2018.

Dimitri P Bertsekas et al. Incremental gradient, subgradient, and proximal methods for convex optimization: A survey. *Optimization for Machine Learning*, 2010(1-38):3, 2011.

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

Guillaume Carlier, Vincent Duval, Gabriel Peyré, and Bernhard Schmitzer. Convergence of entropic schemes for optimal transport and gradient flows. *SIAM Journal on Mathematical Analysis*, 49(2): 1385–1418, 2017.

Peng Chen, Keyi Wu, Joshua Chen, Tom O'Leary-Roseberry, and Omar Ghattas. Projected stein variational newton: A fast and scalable bayesian inference method in high dimensions. *Advances in Neural Information Processing Systems*, 32, 2019.

Tianqi Chen, Emily Fox, and Carlos Guestrin. Stochastic gradient hamiltonian monte carlo. In *International conference on machine learning*, pp. 1683–1691. PMLR, 2014.

Sinho Chewi, Thibaut Le Gouic, Chen Lu, Tyler Maunu, and Philippe Rigollet. Svgd as a kernelized wasserstein gradient flow of the chi-squared divergence. *Advances in Neural Information Processing Systems*, 33:2098–2109, 2020.

Kacper Chwialkowski, Heiko Strathmann, and Arthur Gretton. A kernel test of goodness of fit. In *International conference on machine learning*, pp. 2606–2615. PMLR, 2016.

Gianluca Detommaso, Tiangang Cui, Youssef Marzouk, Alessio Spantini, and Robert Scheichl. A stein variational newton method. *Advances in Neural Information Processing Systems*, 31, 2018.

Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

Andrew Duncan, Nikolas Nüsken, and Lukasz Szpruch. On the geometry of stein variational gradient descent. *arXiv preprint arXiv:1912.00894*, 2019.

Jiaojiao Fan, Amirhossein Taghvaei, and Yongxin Chen. Variational wasserstein gradient flow. *arXiv preprint arXiv:2112.02424*, 2021.

Yihao Feng, Dilin Wang, and Qiang Liu. Learning to draw samples with amortized stein variational gradient descent. *arXiv preprint arXiv:1707.06626*, 2017.

Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pp. 1189–1232, 2001.

Gery Geenens. Curse of dimensionality and related issues in nonparametric functional regression. *Statistics Surveys*, 5:30–43, 2011.

Wenbo Gong, Yingzhen Li, and José Miguel Hernández-Lobato. Sliced kernelized stein discrepancy. *arXiv preprint arXiv:2006.16531*, 2020.

Will Grathwohl, Kuan-Chieh Wang, Jörn-Henrik Jacobsen, David Duvenaud, and Richard Zemel. Learning the stein discrepancy for training and evaluating energy-based models without sampling. In *International Conference on Machine Learning*, pp. 3732–3747. PMLR, 2020.

Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *The Journal of Machine Learning Research*, 13(1):723–773, 2012.

Matthew D Hoffman, Andrew Gelman, et al. The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo. *J. Mach. Learn. Res.*, 15(1):1593–1623, 2014.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Tianyang Hu, Zixiang Chen, Hanxi Sun, Jincheng Bai, Mao Ye, and Guang Cheng. Stein neural sampler. *arXiv preprint arXiv:1810.03545*, 2018.

Michael F Hutchinson. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics-Simulation and Computation*, 18(3):1059–1076, 1989.

Rie Johnson and Tong Zhang. A framework of composite functional gradient methods for generative adversarial models. *IEEE transactions on pattern analysis and machine intelligence*, 43(1):17–32, 2019.

Michael I Jordan, Zoubin Ghahramani, Tommi S Jaakkola, and Lawrence K Saul. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999.

Richard Jordan, David Kinderlehrer, and Felix Otto. The variational formulation of the fokker–planck equation. *SIAM journal on mathematical analysis*, 29(1):1–17, 1998.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

Chang Liu, Jingwei Zhuo, Pengyu Cheng, Ruiyi Zhang, and Jun Zhu. Understanding and accelerating particle-based variational inference. In *International Conference on Machine Learning*, pp. 4082–4092. PMLR, 2019.

Qiang Liu. Stein variational gradient descent as gradient flow. *Advances in neural information processing systems*, 30, 2017.

Qiang Liu and Dilin Wang. Stein variational gradient descent: A general purpose bayesian inference algorithm. *Advances in neural information processing systems*, 29, 2016.

Qiang Liu, Jason Lee, and Michael Jordan. A kernelized stein discrepancy for goodness-of-fit tests. In *International conference on machine learning*, pp. 276–284. PMLR, 2016.

Xing Liu, Harrison Zhu, Jean-Francois Ton, George Wynne, and Andrew Duncan. Grassmann stein variational gradient descent. *arXiv preprint arXiv:2202.03297*, 2022.

Petr Mokrov, Alexander Korotin, Lingxiao Li, Aude Genevay, Justin M Solomon, and Evgeny Burnaev. Large-scale wasserstein gradient flows. *Advances in Neural Information Processing Systems*, 34, 2021.

Atsushi Nitanda and Taiji Suzuki. Gradient layer: Enhancing the convergence of adversarial training for generative models. In *International Conference on Artificial Intelligence and Statistics*, pp. 1008–1016. PMLR, 2018.

Gabriel Peyré. Entropic approximation of wasserstein gradient flows. *SIAM Journal on Imaging Sciences*, 8(4):2323–2351, 2015.

Yuchen Pu, Zhe Gan, Ricardo Henao, Chunyuan Li, Shaobo Han, and Lawrence Carin. Vae learning via stein variational gradient descent. *Advances in Neural Information Processing Systems*, 30, 2017.

Rajesh Ranganath, Sean Gerrish, and David Blei. Black box variational inference. In *Artificial intelligence and statistics*, pp. 814–822. PMLR, 2014.

Adil Salim, Anna Korba, and Giulia Luise. The wasserstein proximal gradient algorithm. *Advances in Neural Information Processing Systems*, 33:12356–12366, 2020.

Santosh Vempala and Andre Wibisono. Rapid convergence of the unadjusted langevin algorithm: Isoperimetry suffices. *Advances in neural information processing systems*, 32, 2019.

Dilin Wang and Qiang Liu. Learning to draw samples: With application to amortized mle for generative adversarial learning. *arXiv preprint arXiv:1611.01722*, 2016.

Dilin Wang, Ziyang Tang, Chandrajit Bajaj, and Qiang Liu. Stein variational gradient descent with matrix-valued kernels. *Advances in neural information processing systems*, 32, 2019.

Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pp. 681–688. Citeseer, 2011.

Jingwei Zhuo, Chang Liu, Jiaxin Shi, Jun Zhu, Ning Chen, and Bo Zhang. Message passing stein variational gradient descent. In *International Conference on Machine Learning*, pp. 6018–6027. PMLR, 2018.

A PROOFS

A.1 PROOF OF EQ. (3)

From the continuity equation (or Fokker-Planck equation), we have

$$\frac{\partial p(t,x)}{\partial t} = -\nabla \cdot [g(t,x)p(t,x)]$$

By chain rule and integration by parts,

$$\begin{aligned}
\frac{dD_{\mathrm{KL}(t)}}{dt} &= \int \frac{\partial p(t,x)}{\partial t}\left(1+\ln\frac{p(t,x)}{p_*(x)}\right)dx \\
&= -\int \nabla\cdot[g(t,x)p(t,x)]\left(1+\ln\frac{p(t,x)}{p_*(x)}\right)dx \\
&= \int [g(t,x)p(t,x)]^\top \nabla \ln\frac{p(t,x)}{p_*(x)}dx \\
&= \int [g(t,x)p(t,x)]^\top \nabla[\ln p(t,x) - \ln p_*(x)]dx \\
&= \int g(t,x)^\top[\nabla p(t,x) - p(t,x)\nabla \ln p_*(x)]dx \\
&= -\int p(t,x)[\nabla\cdot g(t,x) + g(t,x)\nabla_x \ln p_*(x)]dx.
\end{aligned}$$

A.2 DISCUSSION OF EXAMPLE 1

(1) SVGD with linear kernel $k(x,y) = x^\top Ky + 1$

$$g(t,x) = -\Sigma_*^{-1}[(\Sigma_t + (\mu_t-\mu_*)\mu_t^\top)Kx + \mu_t - \mu_*] + Kx. \tag{11}$$

(2) SVGD with RBF kernel $k_\sigma(x,y) = \frac{1}{\sqrt{\sigma^2}}\exp\left(-\frac{1}{2\sigma^2}\|y-x\|^2\right)$

$$g(t,x) = O(x\exp(-\|x\|^2)). \tag{12}$$

(3) Linear function class with L_2 regularization $Q(f(x)) = \frac{1}{2}\mathbf{E}_{p_t}\|f(x)\|^2$.

$$g(t,x) = -\Sigma_*^{-1}(x-\mu_*) + \Sigma_t^{-1}(x-\mu_t). \tag{13}$$

(4) Linear function class with Mahalanobis regularization $Q(f(x)) = \frac{1}{2}\mathbf{E}_{p_t}\|f(x)\|^2_{\Sigma_*^{-1}}$

$$g(t,x) = -(x-\mu_*) + \Sigma_*\Sigma_t^{-1}(x-\mu_t). \tag{14}$$

For optimal transport $g(t,x) = -(x-\mu_*) + \Sigma_t^{-1/2}(\Sigma_t^{1/2}\Sigma_*\Sigma_t^{1/2})^{1/2}\Sigma_t^{-1/2}(x-\mu_t)$.

Proof. We consider these 4 cases seperately.

(1) SVGD with linear kernel.

If $k(x,y) = x^\top y + 1$, then $\nabla_y k(x,y) = Kx$,

$$\begin{aligned}
g(t,x) &= \int p(t,y)(-\Sigma_*^{-1}(y-\mu_*) + \Sigma_t^{-1}(y-\mu_t))(y^\top Kx + 1)dy \\
&= -\int p(t,y)(\Sigma_*^{-1}(y-\mu_*)(y^\top Kx+1))dy \\
&\quad + \int p(t,y)(\Sigma_t^{-1}(yy^\top - \mu_t y^\top))dyKx \\
&= -\Sigma_*^{-1}[(\Sigma_t + (\mu_t-\mu_*)\mu_t^\top)Kx + \mu_t - \mu_*] + Kx
\end{aligned}$$

Note that $\mu_*\mu_t^\top$ is not symmetric, which makes the distribution rotate. We cannot find any K to obtain a proper preconditioning.

(2) SVGD with RBF kernel.

$$
\begin{aligned}
g(t,x) =& \int \frac{1}{\sqrt{2\pi|\Sigma_t|}} \exp\left(-\frac{1}{2}(y-\mu_t)^\top \Sigma_t^{-1}(y-\mu_t)\right) \frac{1}{\sqrt{\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}\|y-x\|^2\right) \\
& \left(\Sigma_*^{-1}(y-\mu_*) - \Sigma_t^{-1}(y-\mu_t)\right) dy \\
=& \int \frac{1}{\sqrt{2\pi|\Sigma_t|}} \exp\left(-\frac{1}{2}\tilde{y}^\top \Sigma_t^{-1}\tilde{y}\right) \frac{1}{\sqrt{\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}\|\tilde{y}+\mu_t-x\|^2\right) \\
& \left(\Sigma_*^{-1}(\tilde{y}+\mu_t-\mu_*) - \Sigma_t^{-1}\tilde{y}\right) d\tilde{y} \\
=& \int \frac{\sqrt{2\pi|(\sigma^{-2}I+\Sigma_t^{-1})^{-1}|}}{\sqrt{2\pi\sigma^2|\Sigma_t|}} \frac{1}{\sqrt{2\pi|(\sigma^{-2}I+\Sigma_t^{-1})^{-1}|}} \exp\left(-\frac{1}{2}(\tilde{y}-\mu_x)^\top \left(\sigma^{-2}I+\Sigma_t^{-1}\right)(\tilde{y}-\mu_x)\right) \\
& \exp\left(\frac{1}{2}(\mu_t-x)^\top \Sigma_t^{-1}(\mu_t-x)\right) \left(\Sigma_*^{-1}(\tilde{y}+\mu_t-\mu_*) - \Sigma_t^{-1}\tilde{y}\right) d\tilde{y} \\
=& \int \frac{1}{\sqrt{\sigma^2|\Sigma_t(\sigma^{-2}I+\Sigma_t^{-1})|}} \tilde{p}(\tilde{y}) \exp\left(\frac{1}{2}(\mu_t-x)^\top \Sigma_t^{-1}(\mu_t-x)\right) \left(\Sigma_*^{-1}(\tilde{y}+\mu_t-\mu_*) - \Sigma_t^{-1}\tilde{y}\right) d\tilde{y} \\
=& C_x \left(\left(\Sigma_*^{-1}-\Sigma_t^{-1}\right)\mu_x + \Sigma_*^{-1}(\mu_t-\mu_*)\right)
\end{aligned}
$$

where

$$
\begin{aligned}
\mu_x &= \sigma^{-2}\left(\sigma^{-2}I+\Sigma_t^{-1}\right)^{-1}(x-\mu_t) = \left(I+\sigma^2\Sigma_t^{-1}\right)^{-1}(x-\mu_t), \\
C_x &= \frac{1}{\sqrt{|\sigma^2 I+\Sigma_t|}} \exp\left(-\frac{1}{2\sigma^2}(x-\mu_t)^\top (I-(I+\sigma^2\Sigma_t^{-1})^{-1})(x-\mu_t)\right)
\end{aligned}
$$

Note that $g(t,x) = O(x\exp(-\|x\|^2))$. When $x \to \infty$, $g(t,x)$ is vanishing.

(3) and (4) are special case of Proposition 2.

Considering the evolution of μ_t, we have

(1) $\frac{d\mu_t}{dt} = -\Sigma_*^{-1}[(\Sigma_t + (\mu_t-\mu_*)\mu_t^\top)K\mu_t + \mu_t - \mu_*] + K\mu_t$., which contains non-symmetric $\mu_*\mu_t^\top$.

(2) The transport is not linear.

(3) $\frac{d\mu_t}{dt} = \Sigma_*^{-1}(\mu_* - \mu_t)$,.

(4) $\frac{d\mu_t}{dt} = \mu_* - \mu_t$, which means that the evolution of (4) is equivalent with optimal transport.

□

A.3 Proof of Proposition 1

Proof. Assume that $Q(f) = \int p(t,x)\|f(x)\|_H^2 dx$, when we have Eq. (4) as

$$
\begin{aligned}
g(t,x) &= \arg\min_{f\in\mathcal{F}} \left[\int p(t,x)[-\nabla\cdot f(x) - f(x)^\top \nabla \ln p_*(x) + \frac{1}{2}\|f(x)\|_H^2]dx\right] && (15) \\
&= \arg\min_{f\in\mathcal{F}} \left[\int p(t,x)[-f(x)^\top \nabla \ln \frac{p_*(x)}{p(t,x)} + \frac{1}{2}\|f(x)\|_H^2]dx\right] && (16) \\
&= \arg\min_{f\in\mathcal{F}} \left[\mathbf{E}_{p_t}[\frac{1}{2}\left\|H^{-1}\nabla \ln \frac{p_*(x)}{p(t,x)}\right\|_H^2 - f(x)^\top \nabla \ln \frac{p_*(x)}{p(t,x)} + \frac{1}{2}\|f(x)\|_H^2]\right] && (17) \\
&= \arg\min_{f\in\mathcal{F}} \left[\frac{1}{2}\mathbf{E}_{p_t}\left\|f(x) - H^{-1}\nabla \ln \frac{p_*(x)}{p(t,x)}\right\|_H^2\right] && (18)
\end{aligned}
$$

□

A.4 PROOF OF PROPOSITION 2

Lemma 1. *(A Variant of Lemma. 11 in Vempala & Wibisono (2019)) Suppose that $p(x) > 0$ is L-smooth. Then, we have*

$$\int p(x) \left\|\nabla \ln p(x)\right\|^2 dx \leq Ld;$$

Proof. By $[\mathbf{A_1}]$, there exists f such that

$$-\int p(t,x)[\nabla \cdot f + f^\top \nabla_x \ln p_*(x)]dx < 0. \tag{19}$$

Thus, consider $f_c(x) = cf(x)$,

$$L(c) = -\int p(t,x)[\nabla \cdot cf + cf^\top \nabla_x \ln p_*(x)]dx =: -C_f c \tag{20}$$

$$l(c) = \int p(t,x)\|cf\|_H^2 dx =: c^2 \int p(t,x)\|f\|_H^2 dx = C_H c^2 \tag{21}$$

where $C_H, C_f > 0$.

When $c = \frac{C_f}{2C_H}$, we have

$$C_H c^2 - C_f c = -\frac{C_f^2}{4C_H} < 0.$$

By choosing $c_0 = \frac{C_f}{2C_H}$,

$$-\int p(t,x)[\nabla \cdot g(t,x) + g(t,x)^\top \nabla \ln p_*(x) + \frac{1}{2}\|g(t,x)\|_H^2]dx \tag{22}$$

$$\leq -\int p(t,x)[\nabla \cdot c_0 f + c_0 f^\top \nabla \ln p_*(x) + \frac{1}{2}\|c_0 f\|_H^2]dx \tag{23}$$

$$< 0 \tag{24}$$

Thus,

$$\frac{dD_{\mathrm{KL}}(t)}{dt} = -\int p(t,x)[\nabla \cdot g(t,x) + g(t,x)^\top \nabla \ln p_*(x)dx < 0$$

By Eq. (3),

$$\frac{dD_{\mathrm{KL}}(t)}{dt} = -\int p(t,x)[\nabla \cdot g(t,x) + g(t,x)^\top \nabla_x \ln p_*(x)]dx \tag{25}$$

$$= -\int p(t,x)[g(t,x)\nabla \ln \frac{p_*(x)}{p(t,x)}]dx \tag{26}$$

$$\geq -\frac{1}{2}\int p(t,x)\left\|\nabla \ln \frac{p_*(x)}{p(t,x)}\right\|^2 dx - \frac{1}{2}\int p(t,x)\left\|g(t,x)\right\|^2 dx \tag{27}$$

$$\geq -(1 + \lambda_{\max}(H)\lambda_{\min}^{-2}(H))\int p(t,x)\left\|\nabla \ln \frac{p_*(x)}{p(t,x)}\right\|^2 dx \tag{28}$$

$$\geq -(1 + \lambda_{\max}(H)\lambda_{\min}^{-2}(H))\left(Ld + \int p(t,x)\left\|\nabla \ln p_*(0) + Lx\right\|^2 dx\right) \tag{29}$$

$$> -\infty \tag{30}$$

where $\int p(t,x)\left\|\nabla \ln \frac{p_*(x)}{p(t,x)}\right\|^2 dx$ can be controlled by Ld ($[\mathbf{A_2}]$ and Lemma 1) and $\mathbf{E}_{p_t}\|x\|^2$.

□

A.5 PROOF OF THEOREM 1

Proof. By μ-log-Sobolev inequality in $[\mathbf{A_4}]$, suppose $g^2 = p_t/p_*$, then we have

$$D_{\mathrm{KL}}(t) \leq \frac{1}{2\mu} \int p(t,x) \left\| \nabla \ln \frac{p(t,x)}{p_*(x)} \right\|^2 dx. \tag{31}$$

When we consider the H-induced norm, for any x, we have $\|x\|_{H^{-1}} \geq \frac{1}{\lambda_{\max}(H)} \|x\|$. Thus,

$$D_{\mathrm{KL}}(t) \leq \frac{\lambda_{\max}(H)}{2\mu} \int p(t,x) \left\| \nabla \ln \frac{p(t,x)}{p_*(x)} \right\|_{H^{-1}}^2 dx. \tag{32}$$

Using Cauchy-Schwarz inequality and $[\mathbf{A_3}]$, we have

$$\begin{aligned}
\frac{dD_{\mathrm{KL}}(t)}{dt} &= -\int p(t,x) \nabla \ln \frac{p_*(x)}{p(t,x)} \cdot g(t,x) dx \\
&= -\int p(t,x) \nabla \ln \frac{p_*(x)}{p(t,x)} \cdot \left(g(t,x) - H^{-1} \nabla \ln \frac{p_*(x)}{p(t,x)} + H^{-1} \nabla \ln \frac{p_*(x)}{p(t,x)} \right) dx \\
&= -\int p(t,x) \left\| \nabla \ln \frac{p(t,x)}{p_*(x)} \right\|_{H^{-1}}^2 dx \\
&\quad + \int p(t,x) \nabla \ln \frac{p_*(x)}{p(t,x)} \cdot \left(g(t,x) - H^{-1} \nabla \ln \frac{p_*(x)}{p(t,x)} \right) dx \\
&\leq -\int p(t,x) \left\| \nabla \ln \frac{p(t,x)}{p_*(x)} \right\|_{H^{-1}}^2 dx \\
&\quad + \int p(t,x) \left\| \nabla \ln \frac{p_*(x)}{p(t,x)} \right\|_{H^{-1}} \left\| g(t,x) - H^{-1} \nabla \ln \frac{p_*(x)}{p(t,x)} \right\|_{H} dx \\
&\leq -\frac{1}{2} \int p(t,x) \left\| \nabla \ln \frac{p(t,x)}{p_*(x)} \right\|_{H^{-1}}^2 dx + \frac{1}{2} \int p(t,x) \left\| g(t,x) - H^{-1} \nabla \ln \frac{p_*(x)}{p(t,x)} \right\|_{H}^2 dx \\
&\leq -\frac{\mu(1-\epsilon)}{\lambda_{\max}(H)} D_{\mathrm{KL}}(t)
\end{aligned}$$

Thus, by Grönwall's inequality,

$$D_{\mathrm{KL}}(t) \leq \exp\left(-\frac{\mu(1-\epsilon)}{\lambda_{\max}(H)} \right) D_{\mathrm{KL}}(0)$$

Remark: Theorem 1 shows that PFG algorithm convergence with linear rate. Moreover, the choice of preconditioning matrix affect the convergence rate with the largest eigenvalue. In real practice, due to the discretized algorithm, one needs to normalize the scaling of H, which is equivalent to the learning rate in practice. For any preconditioning $\tilde{H}$, we need to compute the corresponding learning rate to get $H = \eta^{-1}\tilde{H}$. One usually set that $\|\eta \tilde{H}^{-1} \nabla^2 \ln p_*(x)\|_2^{-1} \leq 1$ to make the discretized (first-order) algorithm feasible ($\eta \leq \|\tilde{H}^{-1} \nabla^2 \ln p_*(x)\|_2$), otherwise the second order error would be too large. The construction is similar to condition number dependency in optimization literature.

Take the fixed largest learning rate for example in the continuous case:

When $\tilde{H} = I$, we have that $\eta = 1/L$, $H = LI$,

$$\frac{dD_{\mathrm{KL}}(t)}{dt} \leq -\frac{1}{2L} \mathbb{E}_{p_t} \left\| \nabla \ln \frac{p_t(x)}{p_*(x)} \right\|^2.$$

When $\tilde{H} = -\nabla^2 \ln p_*$, we have that $\eta = 1$, $H = -\nabla^2 \ln p_*$,

$$\frac{dD_{\mathrm{KL}}(t)}{dt} \leq -\frac{1}{2} \mathbb{E}_{p_t} \left\| \nabla \ln \frac{p_t(x)}{p_*(x)} \right\|_{-(\nabla^2 \ln p_*)^{-1}}^2 \leq -\frac{1}{2L} \mathbb{E}_{p_t} \left\| \nabla \ln \frac{p_t(x)}{p_*(x)} \right\|^2.$$

□

A.6 ANALYSIS OF DISCRETE GAUSSIAN CASE

As is shown in Example 1, given that $x_0 \sim \mathcal{N}(0, I)$ and the target distribution is $\mathcal{N}(\mu_*, \Sigma_*)$, we have the following update for our PFG algorithm,

$$x_{t+1} = x_t + \eta H^{-1}\left(\Sigma_*^{-1}(\mu_* - x_t) + \Sigma_t^{-1}(x_t - \mu_t)\right).$$

By taking the expectation, we have

$$\begin{aligned}\mu_{t+1} - \mu_t &= \eta H^{-1}\left(\Sigma_*^{-1}(\mu_* - \mu_t)\right)\\ \Sigma_{t+1} - \Sigma_t &= \eta H^{-1}\left(\Sigma_t^{-1} - \Sigma_*^{-1}\right)\Sigma_t + \eta\Sigma_t\left(\Sigma_t^{-1} - \Sigma_*^{-1}\right)H^{-1}\\ &\quad + \eta^2 H^{-1}\left(\Sigma_t^{-1} - \Sigma_*^{-1}\right)\Sigma_t\left(\Sigma_t^{-1} - \Sigma_*^{-1}\right)H^{-1}\end{aligned}$$

Since $\Sigma_0 = I$, the eigenvectors of Σ_t will not change during the update. For notation simplicity, we write the diagonal case in the following discussion.

Assume that $\Sigma_* = \mathrm{diag}(\sigma_1^2, \cdots, \sigma_d^2)$, $\Sigma_t = \mathrm{diag}(s_1^2(t), \cdots, s_d^2(t))$, $H = \mathrm{diag}(h_1, \cdots, h_d)$, such that $\sigma_i \geq \sigma_{i+1}$ for any $i = 1, \cdots, d-1$. Without loss of generality, we assume that $\sigma_1 \leq 1$, otherwise we can still rescale the initialization to obtain the similar constraint.

Given the fact that $x - \ln(1+x) \leq \frac{x^2}{2}$ when $x > 0$, we have,

$$D_{\mathrm{KL}}(t) \leq \frac{1}{2}\left(\|\mu_t - \mu_*\|^2_{\Sigma_*^{-1}} + \frac{1}{2}\sum_{i=1}^{d}\left(\frac{s_i^2(t) - \sigma_i^2}{\sigma_i^2}\right)^2\right).$$

Thus, we define

$$C_0 := \frac{1}{2}\left(\|\mu_0 - \mu_*\|^2_{\Sigma_*^{-1}} + \frac{1}{2}\sum_{i=1}^{d}\left(\frac{1 - \sigma_i^2}{\sigma_i^2}\right)^2\right). \tag{33}$$

Considering the i-th dimension (with notation $[\cdot]_i$),

$$[\mu_{t+1}]_i = \left(1 - \frac{\eta}{h_i\sigma_i^2}\right)[\mu_t]_i + \frac{\eta}{h_i\sigma_i^2}[\mu_*]_i \tag{34}$$

$$s_i^2(t+1) = s_i^2(t) + \frac{2\eta s_i^2(t)}{h_i}\left(\frac{1}{s_i(t)^2} - \frac{1}{\sigma_i^2}\right) + \frac{\eta^2 s_i^2(t)}{h_i^2}\left(\frac{1}{s_i(t)^2} - \frac{1}{\sigma_i^2}\right)^2. \tag{35}$$

To guarantee the convergence of the algorithm, we need to make sure that both μ_t and Σ_t converge. By rearranging (34) and (35), we have

$$\frac{[\mu_{t+1}]_i - [\mu_*]_i}{\sigma_i} = \left(1 - \frac{\eta}{h_i\sigma_i^2}\right)\frac{[\mu_t]_i - [\mu_*]_i}{\sigma_i}$$

$$\frac{s_i^2(t+1) - \sigma_i^2}{\sigma_i^2} = \left(1 - \frac{2\eta}{h_i\sigma_i^2}\right)\left(\frac{s_i^2(t) - \sigma_i^2}{\sigma_i^2}\right) + \frac{\eta^2}{h_i^2\sigma_i^2 s_i^2(t)}\left(\frac{s_i^2(t) - \sigma_i^2}{\sigma_i^2}\right)^2.$$

One needs to construct contraction sequences of both mean and variance gap, which are guided by $1 - \eta/(h_i\sigma_i^2)$ and $1 - 2\eta/(h_i\sigma_i^2)$. Notice that $s_i^2(0) \geq \sigma_i^2$. Assume that $\eta \leq \min_i h_i\sigma_i^2/2$. Then we have $s_i^2(t+1) \geq \sigma_i^2$.

Thus, we obtain

$$\frac{\eta}{h_i}\left|\frac{1}{\sigma_i^2} - \frac{1}{s_i^2(t)}\right| \leq 1.$$

It is natural to obtain the contraction of both mean and variance gap,

$$\|\mu_{t+1} - \mu_*\|^2_{\Sigma_*^{-1}} \leq \max_i\left(1 - \frac{\eta}{h_i\sigma_i^2}\right)^2\|\mu_t - \mu_*\|^2_{\Sigma_*^{-1}}$$

$$\left|\frac{s_i^2(t+1)-\sigma_i^2}{\sigma_i^2}\right| \leq \left(1-\frac{\eta}{h_i\sigma_i^2}\right)\left|\frac{s_i^2(t)-\sigma_i^2}{\sigma_i^2}\right|$$

$$D_{\mathrm{KL}}(t) \leq \max_i \left(1-\frac{\eta}{h_i\sigma_i^2}\right)^{2t} C_0,$$

where C_0 is defined in Eq. (33).

When $h_i = 1$, we have $\eta = \frac{\sigma_d^2}{2}$

$$D_{\mathrm{KL}}(t) \leq \left(1-\frac{\sigma_d^2}{2\sigma_1^2}\right)^{2t} C_0.$$

When $h_i = \sigma_i^{-2}$, we have $\eta = \frac{1}{2}$

$$D_{\mathrm{KL}}(t) \leq \frac{1}{4^t} C_0.$$

This result is equivalent to the Remark in A.5.

A.7 Proof of Infinite-dimensional case of RKHS

Proof. Assume the feature map, $\psi(x) : \mathbb{R}^d \to \mathcal{H}$ and let kernel $k(x,y) = \langle \psi(x), \psi(y)\rangle_{\mathcal{H}}$, $Q(f) = \frac{1}{2}\|f\|_{\mathcal{H}^d}$. One can perform spectral decomposition to obtain that

$$k(x,y) = \sum_{i=1}^{\infty} \lambda_i \psi_i(x)\psi_i(y)$$

where $\psi_i : \mathbb{R}^d \to \mathbb{R}$ are orthonormal basis and λ_i is the corresponding eigenvalue.

For any $g \in \mathcal{H}^d$, we have the decomposition,

$$g(x) = \sum_{i=1}^{\infty} g_i \sqrt{\lambda_i}\psi_i(x),$$

where $g_i \in \mathbb{R}^d$ and $\sum_{i=1}^{\infty} \|g_i\|^2 < \infty$.

The solution is defined by

$$\hat{g} = \arg\min_{g\in\mathcal{H}^d} -\int p(t,x)\left(\nabla\cdot g + \nabla \ln p_*(x)^\top g(x)\right) + \frac{1}{2}\|g\|_{\mathcal{H}^d}^2, \tag{36}$$

$$= \arg\min_{g\in\mathcal{H}^d} -\int p(t,x)\left(\nabla\cdot \sum_{i=1}^{\infty} g_i\sqrt{\lambda_i}\psi_i(x) + \sum_{i=1}^{\infty}\sqrt{\lambda_i}\nabla\ln p_*(x)^\top g_i\psi_i(x)\right) + \sum_{i=1}^{\infty}\|g_i\|^2, \tag{37}$$

which gives

$$\hat{g}_i = \sqrt{\lambda_i}\int p(t,y)[\nabla\psi_i(y) + \nabla\ln p_*(y)\psi_i(y)]dy.$$

This implies that

$$g(t,x) = \sum_{i=1}^{\infty}\sqrt{\lambda_i}\hat{g}_i\psi_i(x) = \int p(t,y)[\nabla_y k(y,x) + \nabla\ln p_*(y)k(y,x)]dy, \tag{38}$$

which is equivalent to SVGD.

□

B MORE DISCUSSIONS

B.1 SCORE MATCHING

Score matching (SM) is related to our algorithm due to the integration by parts technique, which uses parameterized models to estimate Stein score $\nabla \ln p_*$.

We made some modifications to the original techniques in SM: (1) We extend the score matching $\nabla \ln p_*$ to Wasserstein gradient approximation $\nabla \ln p_* - \nabla \ln p_t$; (2) We introduce the geometry-aware regularization to approximate the preconditioned gradient. Thus, our proposed modifications make it suitable for sampling tasks.

In a word, both SM and PFG are derived from the integration by parts technique. SM is a very promising framework in generative modeling and our PFG is more suitable for sampling tasks.

B.2 COMPARISON WITH JKO-BASED WASSERSTEIN GRADIENT FLOW

There is another line of work to approximate Wasserstein gradient flow through JKO-discretization Mokrov et al. (2021); Alvarez-Melis et al. (2021); Fan et al. (2021). The continuous versions of these Wasserstein gradient flow algorithms are related to our algorithm when the functional is chosen as KL divergence. These are promising alternative algorithms to PFG in transport tasks. From the theoretical view, they solves the JKO operator with special neural networks (ICNN) and aims to estimate the Wasserstein gradient of general functionals, including KL/JS divergence, etc. On the other hand, our algorithm is motivated by the continuous case of KL Wasserstein gradient flow and we only consider the Euler discretization (rather than JKO), which is the setting of particle-based variational inference.

When we consider the KL divergence and posterior sampling task, our proposed method is more efficient than Wasserstein gradient flow based ones, due to the flexibility of the gradient estimation function. We provide the empirical results on Bayesian logistic regression below (Covtype dataset).

Table 4: Bayesian logistic regression (Covtype dataset)

METHOD	ACCURACY (%)	NLL	TIME (S)
OURS	**75.7**	**0.58**	**0.2**
JKO-ICNN	75.0	0.62	53.2
JKO-VARIATIONAL	75.3	0.59	29.1

According to the table, our algorithm outperforms JKO-type Wasserstein gradient methods. Moreover, it is important to mention that JKO-type Wasserstein gradient is extremely computation-exhausted, due to the computation of JKO operator. Thus, it would be more challenging to apply them to larger tasks such as Bayesian neural networks.

B.3 VARIANCE COLLAPSE AND HIGH-DIMENSIONAL PERFORMANCE

Some recent works Ba et al. (2021); Gong et al. (2020); Zhuo et al. (2018); Liu et al. (2022) suggest that SVGD suffers from curse of dimensionality and the variance of SVGD-trained particles tend to diminish in high-dimensional spaces. We highlight that one the key issues is the capacity of RBF function space is misspecified. As shown in Tab. 5, when fitting a Gaussian distribution, an ideal function class is the linear function class given the Gaussian initialization. However, provided with RBF-based SVGD, the variance collapse phenomenon is extremely severe. On the contrary, when using a proper function class (linear), both SVGD and PFG performs well. More importantly, for some powerful non-linear function classes (neural networks), although it is not well-specified one, it still performs well due to the approximation capacity. This indicates that when the target distribution is more complex (the well-specified function class is not simply obtained), the effectiveness of neural network function class would be particularly important.

Table 5: Illustration of Variance Collapse due to the misspecified function classes (20-dim $\mathcal{N}(0, I)$)

METHOD	TARGET	SVGD (RBF)	SVGD (LINEAR)	PFG (LINEAR)	PFG (NEURAL NETWORK)
VARIANCE	1.00	0.35	1.01	1.00	0.99

To further justify the effectiveness of PFG algorithm in high dimensional cases, we evaluate our model under different settings following Liu et al. (2022). Tab. 6 shows that our PFG algorithm is robust to high dimensional fitting problems, which is comparable to GSVGD Liu et al. (2022) in Gaussian case.

Table 6: Estimating the dimension-averaged marginal variance of $\mathcal{N}(0, I)$

DIMENSION	20	40	60	80	100
SVGD	0.35	0.18	0.12	0.09	0.04
GSVGD	0.96	0.97	**1.00**	1.01	**1.00**
PFG	**1.00**	**0.99**	0.98	**1.00**	0.97

We also include a further justification in Tab. 7, which compute the energy distance between particle the target distribution and the particle estimation on a 4-mode Gaussian mixture distribution. In this case, the function class of gradient should be much larger than linear case. We can find that the resulting performance improves the previous work (GSVGD).

Table 7: Energy distance ($\times 10^{-2}$) between the target distribution and the particle estimation on Multimodal Gaussian mixture distribution (4 modes)

DIMENSION	20	40	60	80	100
SVGD	2.1	6.7	10.8	24.9	39.6
GSVGD	1.0	2.3	3.4	3.2	3.5
PFG	**0.6**	**0.8**	**1.2**	**1.6**	**2.1**

B.4 OTHER PRECONDITIONED PARTICLE-BASED ALGORITHMS

Stein variational Newton method (SVN) Detommaso et al. (2018) approximates the Hessian to accelerate SVGD. However, due to the gap between SVGD and Wasserstein gradient flow, the theoretical interpretation is not clear. Matrix SVGD Wang et al. (2019) leverages more general matrix-valued kernels in SVGD, which includes SVN as a variant and is better than SVN. Chen et al. (2019) perform a parallel update of the parameter samples projected into a low-dimensional subspace by an SVN method. It implements dimension reduction to SVN algorithm. Liu et al. (2022) projects SVGD onto arbitrary dimensional subspaces with Grassmann Stein kernel discrepancy, which implement Riemannian optimization technique to find the proper projection space.

Note that all of these algorithms are based on RKHS norm regularization/SVGD. As we mentioned in Example 1, due to the introduction of RKHS, the preconditioning matrix is often hard to find. For example, the preconditioning matrix of the linear kernel varies in time, while our algorithm only needs a constant matrix. For RBF kernel, the previous works have demonstrated the improvement of the Hessian inverse matrix, but the design is still heuristic due to the gap between the Wasserstein gradient and SVGD. Our algorithm directly approximates the preconditioned Wasserstein gradient, and further analysis is clear and conclusive. Also, the SVGD-based algorithms suffer from the drawbacks of RKHS norm regularization, as discussed in Example 1.

B.5 MORE COMPARSIONS

We also include a more complete empirical results on Bayesian neural networks in Tab. 8, including the comparison with Matrix SVGD (M-SVGD) Wang et al. (2019) and Sliced SVGD (S-SVGD) Gong et al. (2020). Our algorithm is still the most competitive one according to the table.

Table 8: Averaged test root-mean-square error (RMSE) and test log-likelihood of Bayesian Neural Networks on UCI datasets (100 particles). Results are computed by 10 trials.

	AVERAGE TEST RMSE			AVERAGE TEST LOG-LIKELIHOOD		
DATASET	M-SVGD	S-SVGD	PFG	M-SVGD	S-SVGD	PFG
BOSTON	$2.72_{\pm 0.16}$	$2.87_{\pm 0.16}$	$\mathbf{2.47}_{\pm 0.11}$	$-2.86_{\pm 0.14}$	$-2.50_{\pm 0.16}$	$\mathbf{-2.35}_{\pm 0.12}$
CONCRETE	$\mathbf{4.72}_{\pm 0.11}$	$4.88_{\pm 0.08}$	$4.74_{\pm 0.14}$	$-3.21_{\pm 0.07}$	$-3.00_{\pm 0.02}$	$\mathbf{-2.83}_{\pm 0.16}$
ENERGY	$0.87_{\pm 0.03}$	$1.13_{\pm 0.05}$	$\mathbf{0.48}_{\pm 0.04}$	$\mathbf{-1.25}_{\pm 0.04}$	$-1.54_{\pm 0.05}$	$-1.28_{\pm 0.06}$
WINERED	$0.64_{\pm 0.01}$	$0.64_{\pm 0.01}$	$\mathbf{0.60}_{\pm 0.02}$	$-1.63_{\pm 0.02}$	$-1.82_{\pm 0.06}$	$\mathbf{-1.61}_{\pm 0.03}$
WINEWHITE	$0.62_{\pm 0.01}$	$0.62_{\pm 0.01}$	$\mathbf{0.59}_{\pm 0.02}$	$-1.65_{\pm 0.03}$	$-1.71_{\pm 0.03}$	$\mathbf{-1.58}_{\pm 0.04}$

C IMPLEMENTATION DETAILS

All experiments are conducted on Python 3.7 with NVIDIA 2080 Ti. Particularly, we use PyTorch 1.9 to build models. Besides, Numpy, Scipy, Sklearn, Matplotlib, Pillow are used in the models.

C.1 SYNTHETIC EXPERIMENTS

C.1.1 ILL-CONDITIONED GAUSSIAN DISTRIBUTION

For ill-conditioned Gaussian distribution, we reproduce the continuous dynamics with Euler discretization. We consider 4 cases: (a) $\mu_* = [1, 1]^\top$, $\Sigma_* = \mathrm{diag}(1, 1)$; (b) $\mu_* = [1, 1]^\top$, $\Sigma_* = \mathrm{diag}(100, 1)$; (c) $\mu_* = [20, 20]^\top$, $\Sigma_* = \mathrm{diag}(1, 1)$; (d) $\mu_* = [20, 20]^\top$, $\Sigma_* = \mathrm{diag}(100, 1)$ to illustrate the behavior of SVGD and our algorithm clearly.

Notably, for our algorithms and SVGD (linear kernel), we use step size 10^{-3} to approximate the continuous dynamics and solve the mean and variance exactly (equivalent to infinite particles); for SVGD with RBF kernel, we use step size 10^{-2} with $1,000$ particles to approximate mean and variance. For the neural network function class, we assume that the function class of two-layer neural network $\mathcal{F}_\theta = \{f_\theta : \theta \in \Theta\}$. In practice, we may incorporate base shift f_0 as gradient boosting to accelerate the convergence, we use $\mathcal{F} = \{f_0 + f_\theta : f_\theta \in \mathcal{F}_\theta\}$, where $f_0(x) = 0$ or $f_0(x) = c\nabla \ln p_*(x)$ with some constant c. In practice, we select c from $\{0, 0.1, 0.2, 0.5, 1\}$ by validation. Empirically, high dimensional models (such as Bayesian neural networks) can be accelerated significantly with the base function.

C.1.2 GAUSSIAN MIXTURE

For Gaussian Mixture, we sample a 2-d 10-cluster Gaussian Mixture, where cluster means are sampled from standard Normal distribution, covariance matrix is $0.1^2 I$, the marginal probability of each cluster is $1/10$. The function class is 2-layer network with 32 hidden neurons and Tanh activation function. For inner loop, we choose $T' = 5$ and SGD optimizer (lr = 1e-3) with momentum 0.9. For particle optimization, we choose step-size = 1e-1. For SVGD, we choose RBF kernel with median bandwidth, step-size is 1e-2.

C.2 APPROXIMATE POSTERIOR INFERENCE

To compute the trace, we use the randomized trace estimation (Hutchinson's trace estimator) Hutchinson (1989) to accelerate the computation .

- Sample $\{\xi_i\}_{i=1}^K$, such that $\mathbf{E}\xi = 0$ and $\mathbf{Cov}\xi = I$;
- Compute $\hat{L}(\theta) = \frac{1}{m}\sum_{i=1}^m \Big(Q(f_\theta(x_t^i) + f_0(x_t^i)) - f_\theta(x_t^i)\cdot\nabla U(x_t^i) - \frac{1}{K}\sum_{j=1}^K \xi_i^\top \nabla f_\theta(x_t^i)\xi_i\Big)$;

- Compute $\hat{L}(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(Q(f_\theta(x_t^i) + f_0(x_t^i)) - f_\theta(x_t^i)\cdot\nabla U(x_t^i) - \nabla\cdot f_\theta(x_t^i)\right)$;
- Update $\theta_t^{t'} = \theta_t^{t'-1} - \eta'\nabla\hat{L}(\theta_t^{t'-1})$;

where ξ is chosen as Rademacher distribution.

C.2.1 LOGISTIC REGRESSION

For logistic regression, we compare our algorithm with SVGD in terms of the "goodness" of particle distribution. The ground truth is computed by NUTS (Hoffman et al., 2014). The mean is computed by 40,000 samples, the MMD is computed with 4,000 samples. The H in this part is chosen as $H = I$ or $H = \hat{H}^{0.1}$.

In the experiments, we select step-size from $\{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$ for each algorithm by validation. And we use Adam optimizer without momentum. For SVGD, we choose RBF kernel with median bandwidth. For PFG, we select inner loop from $\{1, 2, 5, 10\}$ by validation. The hidden neuron is 32.

C.2.2 HIERARCHICAL LOGISTIC REGRESSION

For hierarchical logistic regression, we use the test performance to measure the different algorithms, including likelihood and accuracy. The H in this part is chosen as $H = \hat{H}^{0.5}$.

For all experiments, batch-size is 200 and we select step-size from $\{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$ for each algorithm by validation. And we use Adam optimizer without momentum. For SVGD, we choose RBF kernel with median bandwidth. For PFG, we select inner loop from $\{1, 2, 5, 10\}$ by validation. The hidden neuron is 32.

C.2.3 BAYESIAN NEURAL NETWORKS (BNN)

We have two experiments in this part: UCI datasets, MNIST classification. The metric includes MSE/accuracy and likelihood. The hidden layer size is chosen from $\{32, 64, 128, 256, 512\}$. To approximate H, we use the approximated diagonal Hessian matrix $\hat{H}$, and choose $H = \hat{H}^\alpha$, where $\alpha \in \{0, 0.1, 0.2, 0.5, 1\}$. The inner loop T' of PFG is chosen from $\{1, 2, 5, 10\}$ and with SGD optimizer (lr = 1e-3, momentum=0.9).

For UCI datasets, we follow the setting of SVGD Liu & Wang (2016). Data samples are randomly partitioned to two parts: 90% (training), 10% (testing) and we use 100 particles for inference. We conduct the two-layer BNN with 50 hidden units (100 for Year dataset) with ReLU activation function. The batch-size is 100 (1000 for Year dataset). Step-size is 0.01.

For MNIST, we conduct the two-layer BNN with 128 neurons. We compared the experiments with different particle size. Step-sizes are chosen from $\{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$. The batch-size is 100.