3. Algorithms

As stated in subsection 2.4, the core concept of Bayesian Machine Learning is to sample from the the posterior distribution of weights $p(\boldsymbol{\theta}|D)$ (or $p(\boldsymbol{\theta}, \boldsymbol{\lambda}|D)$, in the case of the inverse problem).
However, the exact reconstruction of the posterior is unfeasible, as witnessed by the necessity to approximate the Bayes theorem only with the proportionality relation in Equation 20.

In this chapter, we propose the three algorithms implemented that represent different solutions to sample $\boldsymbol{\theta}$ values from the a distribution approximating the real one reasonably well:

- **Hamiltonian Monte Carlo (HMC)**, a common Markov Chain Monte Carlo method which exploits a discrete simulation of Hamiltionian Dynamics and a correction with an acceptance-rejection step;
- **Variational Inference (VI)**, that aims at reconstructing a surrogate for the target distribution by looking for its best approximation within a parametric family that is simpler than the unknown target;
- **Stein Variational Gradient Descent (SVGD)**, that is based on a similar philosophy of VI and is the counterpart of gradient descent which iteratively transports a set of "particles" to match the target.

We will devote to each of them a specific section covering theoretical foundations and algorithm mechanisms.

Log-posterior

In these algorithms we operate with the logarithm of the posterior, hence we devote this section to the computation of this quantity that will be required later on.
With the rule of the logarithm of the product, the relation in Equation 21 becomes:

$$L(\boldsymbol{\theta}) := \log(p(\boldsymbol{\theta}|D,R)) \propto \log(p(D,R|\boldsymbol{\theta})) + \log(p(\boldsymbol{\theta})) \tag{26}$$

By substituting 22 and 15 in Equation 26, we get

$$L(\boldsymbol{\theta}) \propto \log(p(D_u|\boldsymbol{\theta})) + \log(p(D_b|\boldsymbol{\theta})) + \log(p(D_f|\boldsymbol{\theta})) + \log(p(R|\boldsymbol{\theta})) + \log(p(\boldsymbol{\theta})) \tag{27}$$

Thanks to the exponential nature of the terms, the above relation can be reduced to a sum of mean squared errors. Indeed, from 16–18 and 23 we get

$$\log(p(D_u|\boldsymbol{\theta})) \propto -\frac{1}{2\sigma_u^2}\sum_{i=1}^{N_u}(\tilde{\mathbf{u}}^{(i)} - \overline{\mathbf{u}}^{(i)})^2 + \frac{N_u}{2}\log\left(\frac{1}{\sigma_u^2}\right), \tag{28}$$

$$\log(p(D_b|\boldsymbol{\theta})) \propto -\frac{1}{2\sigma_b^2}\sum_{i=1}^{N_b}(\tilde{\mathbf{u}}^{(i)} - \overline{\mathbf{u}}^{(i)})^2 + \frac{N_b}{2}\log\left(\frac{1}{\sigma_b^2}\right), \tag{29}$$

$$\log(p(D_f|\boldsymbol{\theta})) \propto -\frac{1}{2\sigma_f^2}\sum_{i=1}^{N_f}(\tilde{\mathbf{f}}^{(i)} - \overline{\mathbf{f}}^{(i)})^2 + \frac{N_f}{2}\log\left(\frac{1}{\sigma_f^2}\right), \tag{30}$$

$$\log(p(R|\boldsymbol{\theta})) \propto -\frac{1}{2\sigma_r^2}\sum_{i=1}^{N_{col}}(\mathcal{R}(\tilde{\mathbf{u}}^{(i)}) - 0)^2 + \frac{N_{col}}{2}\log\left(\frac{1}{\sigma_r^2}\right). \tag{31}$$

In [3], Equation 27 is seen as a linear combination of the terms with coefficients $\{\alpha_i\}_{i=1}^5$:

$$L(\boldsymbol{\theta}) \propto \alpha_1\log(p(D_u|\boldsymbol{\theta})) + \alpha_2\log(p(D_b|\boldsymbol{\theta})) + \alpha_3\log(p(D_f|\boldsymbol{\theta})) + \alpha_4\log(p(R|\boldsymbol{\theta})) + \alpha_5\log(p(\boldsymbol{\theta}))$$

However, in this work we excluded this option for two main reasons:

1. Due to the removal of the denominator in the application of the Bayes's Theorem, Equation 20 cannot represent a probability distribution anymore, but only an approximation. However, the introduction of coefficients in Equation 27 would still imply a further dissociation from the Bayes's Theorem.
2. The goal of give a different weight to each term can still be reached by remaining coherent with the probabilistic interpretation, thanks to the user's choice of the uncertainties.

3.1. Hamiltonian Monte Carlo

The first method we propose is Hamiltionian Monte Carlo, to whom we will refer as HMC; by integrating Monte Carlo techniques and Hamiltonian Dynamics, it produces a sequence of random samples which converge to being distributed according to our target probability distribution.

3.1.1. Theoretical Foundations

This algorithm is a classic in the family of Markov Chain Monte Carlo (MCMC) methods, which enable sampling from a probability distribution by constructing a Markov chain that has the desired distribution as its equilibrium distribution. In HMC, the purpose consists in obtaining a sequence of random samples which converge to being distributed according to a target probability distribution for which direct sampling is difficult; to do so, we first simulate an Hamiltonian dynamics using numerical integration, and then correct with an acceptance-rejection step, to reduce the discretization error.

Hamiltonian Dynamics

Let us consider the logarithm of the posterior in 27, and define a *potential energy* as its opposite:

$$U(\boldsymbol{\theta}) \coloneqq -L(\boldsymbol{\theta})$$

Note that this relationship in terms of sign between potential energy and (log) posterior is meaningful: the natural goal in this setting is indeed maximizing the probability and, by following the usual physical convention, minimizing the potential energy.
Then, we use it to define the Hamiltonian function, that will describe the dynamics of a continuum system; note that a term which mimics kinetic energy is added as well:

$$H(\boldsymbol{\theta}, \mathbf{r}) \coloneqq U(\boldsymbol{\theta}) + \frac{1}{2}\mathbf{r}^T\mathbf{M}^{-1}\mathbf{r}, \tag{32}$$

in the above formula, $\mathbf{r}$ is an auxiliary momentum variable and $\mathbf{M}$ plays the role of a mass matrix; it is common to set $\mathbf{M} = \mathbf{I}$, or $\mathbf{M} = \alpha\mathbf{I}$, for some $\alpha > 0$.
The Hamiltonian system associated to Equation 32 reads as

$$\begin{cases} \dfrac{d\boldsymbol{\theta}}{dt} = \dfrac{dH}{d\mathbf{r}} = \mathbf{M}^{-1}\mathbf{r} \\ \dfrac{d\mathbf{r}}{dt} = -\dfrac{dH}{d\boldsymbol{\theta}} = -\nabla U(\boldsymbol{\theta}). \end{cases} \tag{33}$$

Now, we establish a link between Hamiltonian dynamics and probabilities: basically, solving the system in 33 corresponds to generate samples of $(\boldsymbol{\theta}, \mathbf{r})$ from the joint distribution

$$\pi(\boldsymbol{\theta}, \mathbf{r}) \sim \exp(-H(\boldsymbol{\theta}, \mathbf{r})) \tag{34}$$

and the sampling from the above is used to mimic the sampling from the posterior distribution $p(\boldsymbol{\theta}|D, R)$, which - up to a constant - coincides with $\exp(-U(\boldsymbol{\theta}))$.
As we are not interested in the samples of $\mathbf{r}$, we then discard them and consider only the samples of $\boldsymbol{\theta}$, which follow the marginal distribution $p(\boldsymbol{\theta}|D, R)$.

Leap Frog

For what concerns the solution of Equation 33, we proceed numerically with the *leap-frog* method, consisting in introducing L additional steps between subsequent updates of $\boldsymbol{\theta}$ of the form:

$$\text{for } k = 1, \dots, L$$

$$\begin{cases} \mathbf{r}^{(k+\frac{1}{2})} = \mathbf{r}^{(k)} - \dfrac{\Delta t}{2}\nabla(U(\boldsymbol{\theta}^{(k)})) \\ \boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} + \Delta t\mathbf{M}^{-1}\mathbf{r}^{(k+\frac{1}{2})} \\ \mathbf{r}^{(k+1)} = \mathbf{r}^{(k+\frac{1}{2})} - \dfrac{\Delta t}{2}\nabla(U(\boldsymbol{\theta}^{(k+1)})). \end{cases} \tag{35}$$

In Equation 35, Δt denotes the step size, and it should be related to L, which represents the number of steps to perform for each update, as we will illustrate in subsubsection 3.1.3.

Acceptance-Rejection

In order to reduce the error introduced with the discretization, the new values are not always accepted, and this choice is controlled in the following acceptance-rejection step.
Considering two subsequent values of $\boldsymbol{\theta}$, that will be denoted by $\boldsymbol{\theta}^{(0)}$ and $\boldsymbol{\theta}^{(1)}$ for simplicity, we compute α:

$$\alpha \coloneqq \min\left(1, \frac{\exp(-H(\boldsymbol{\theta}^{(1)}, \mathbf{r}^{(1)}))}{\exp(-H(\boldsymbol{\theta}^{(0)}, \mathbf{r}^{(0)}))}\right) = \min(1, \exp(-(H(\boldsymbol{\theta}^{(1)}, \mathbf{r}^{(1)}) - H(\boldsymbol{\theta}^{(0)}, \mathbf{r}^{(0)})))).$$

This value is the probability of acceptance of $\boldsymbol{\theta}^{(1)}$: we accept the sampled value $\boldsymbol{\theta}^{(1)}$ if $\alpha \geq p$ with $p \sim \mathcal{U}(0,1)$. Defining $h \coloneqq H(\boldsymbol{\theta}_1, \mathbf{r}_1) - H(\boldsymbol{\theta}_0, \mathbf{r}_0)$, we can rewrite the expression of the acceptance rate as:

$$\alpha = \min(1, \exp(-h)) \Longrightarrow h < 0 \Rightarrow \alpha = 1$$

When $h < 0$, it means that the Hamiltonian has decreased, hence we would like to accept always the new parameters. By contrast, when $h > 0$ the Hamiltonian get worse and we would like to accept with a probability decreasing with h. The overall process outputs a sequence of network parameters $\{\boldsymbol{\theta}^{(i)}\}_{i=1}^{N}$; not all of them are taken into account, but:

- we discard some initial samples, introducing a *burn-in*, to gain in the quality of the final prediction;
- we introduce a *stride*, and consider in the final list of $\boldsymbol{\theta}$s one parameter set each S. In this way, we reduced the correlation that could arise between subsequent samples, and improving the assumption that the final sampling of the parameters is independent and identically distributed.

3.1.2. Algorithm

We now report the complete algorithm that has been implemented:

Algorithm 4: Hamiltonian Monte Carlo

```
Initialization:
    $\boldsymbol{\theta}^{(0)}$ initial value of $\boldsymbol{\theta}$
    $N$ total number of iterations
    $B$ burn-in (number of initial samples to exclude)
    $L$ number of leap-frog steps
    $S$ stride
    $\Delta t$ step size
    $\alpha$ for defining the mass matrix $\mathbf{M} = \alpha\mathbf{I}$
for $k = 1, \dots, N$ do
    Leap-frog step:
    sample $\mathbf{r}^{(k-1)} \sim \mathcal{N}(0, \mathbf{M})$
    set $(\boldsymbol{\theta}_0, \mathbf{r}_0) = (\boldsymbol{\theta}^{(k-1)}, \mathbf{r}^{(k-1)})$
    for $i = 0, \dots, (L-1)$ do
        $\mathbf{r}_i = \mathbf{r}_i - \frac{\Delta t}{2}\nabla U(\boldsymbol{\theta}_i)$
        $\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i + \Delta t \mathbf{M}^{-1}\mathbf{r}_i$
        $\mathbf{r}_{i+1} = \mathbf{r}_i - \frac{\Delta t}{2}\nabla U(\boldsymbol{\theta}_{i+1})$
    Acceptance-Rejection step:
    sample $p \sim \mathcal{U}(0,1)$
    $\alpha = \min(1, \exp(-(H(\boldsymbol{\theta}_L, \mathbf{r}_L) - H(\boldsymbol{\theta}_0, \mathbf{r}_0))))$
    if $p \geq \alpha$ then
        $\boldsymbol{\theta}^{(k)} = \boldsymbol{\theta}_L$
    else
        $\boldsymbol{\theta}^{(k)} = \boldsymbol{\theta}^{(k-1)}$
From the set $\{\boldsymbol{\theta}^{(i)}\}_{i=1}^{N}$, extract a subset $\{\boldsymbol{\theta}_i^*\}_{i=1}^{(N-B)/S}$ using
    Burn-in: discard the first $B$ samples
    Stride: take one sample each $S$, starting from the final one and going backwards
```

Then, the set of weights is used to compute samples of the solution $\mathbf{u}$ and of the parametric field $\mathbf{f}$ by producing neural network outputs setting each element of $\{\boldsymbol{\theta}_i^*\}_i$ as network parameters, and obtaining the sets $\{\tilde{\mathbf{u}}(\boldsymbol{\theta}_i^*)\}_i$ and $\{\tilde{\mathbf{f}}(\boldsymbol{\theta}_i^*)\}_i$. Thanks to statistics of these samples, we are able to quantify uncertainty and to produce a confidence interval for the network output, after having computed mean and standard deviations of the samples.

3.1.3. Parameters' choice

The choice of the number of epochs N or of the burn-in B has been performed with fine-tuning; we set N to a value able to reconstruct with a fairly low error the functions in a suitable computational time (no simulation with HMC training exceeded 20 min) and to keep the acceptance rate above 80%: such value is a trade-off between computational efficiency and guarantee of being close to the optimal solution.

For what concerns B, we always kept at least the 50% of the samples. Note that in our implementation the choice of this value has consequences on two different aspects: on one hand, indeed, we use the burn-in to discard the initial samples which usually are not meaningful and affected by the model initialization. To perform the same task without discarding samples we implemented also another possibility, that is letting HMC start after some Adam iterations: in this way the network parameters at the beginning of the bayesian algorithm are closer to the optimal ones and not completely random.

On the other hand, we let B determine the entrance of the information coming from physics into the log posterior; indeed, we noticed that performing differentiation on the network output when it was far from its true value was harmful for the learning process. The contribution of the physical loss is indeed in general heavier than the one of the fitting loss, given the computational burden of automatic differentiation. Therefore, the strategy that turned out to be optimal is to first get closer to the output only with the fitting points, then let the PDE come into play and improve the reconstruction – or reconstruct the part of the output which was lacking of fitting points.

At the current status, there is no strategy in literature for fixing the optimal values of L and Δt. A straightforward idea is to establish an inverse proportionality relation between them, by fixing their product to be equal to a constant K:

$$K = L\Delta t.$$

In this way indeed, we can fix the total "distance" travelled, but covering it in more smaller steps is clearly computationally more demanding. Hence, once K is chosen, the general rule to be followed is to try to reduce L the most possible in favour of bigger Δt (that does not affect computational time), but retaining a fairly high acceptance rate. The risk of big Δt is indeed a drop in performance, as we remarked that the update of the network parameters needed to be gradual to prevent turning to a regime in which the new proposal is always rejected (due to the explosion of the gradients in Equation 35).

3.2. Variational Inference

We will illustrate in this section another two methods which fall into a different framework: the one of Variational Bayesian Methods, and we first present the Variational Inference (VI) method.
In the purpose of approximating a posterior probability, variational inference-like methods are an alternative to Monte Carlo sampling methods for taking a fully Bayesian approach to statistical inference over complex distributions that are difficult to evaluate directly or sample.

In particular, Monte Carlo techniques can be slow and have difficulties in reaching the convergence. In variational methods we changed perspective, because we do not provide an indirect approximation of the posterior through a set samples, but we obtain a locally-optimal, exact analytical solution for an approximation of the posterior from which samples can later be generated.

3.2.1. Theoretical Foundations

The family of Variational Inference methods is based on the strategy to turn the problem (in this case, the sampling of $\boldsymbol{\theta}$s from a distribution) into a deterministic optimization that approximates the target distribution $P(\boldsymbol{\theta}|D)$ with a simpler distribution $Q(\boldsymbol{\theta}|\zeta)$.

Distance of Probabilities

To perform this step, we need to establish how accurate is the approximation of a probability measure, hence a tool which captures similarities between probability distributions is needed. We use the Kullback–Leibler (KL) divergence between the target distribution and the one we are reconstructing: the KL divergence is a *statistical distance*, and lets us capture how much one probability distribution is similar to another.

Definition 3.1 (Kullback–Leibler divergence). *Given a measure space (X, μ) and two probability distributions P and Q with densities, respectively, $p(x)d\mu$ and $q(x)d\mu$, the KL divergence between P and Q is defined as*

$$\mathcal{D}_{KL}(P||Q) = \int_X p(x) \log\left(\frac{p(x)}{q(x)}\right) d\mu$$

Then, algorithms of this family try to minimize the KL divergence between the target and the approximated distribution. In our context of B-PINNs we are interested in the value of ζ that minimizes $\mathcal{D}_{KL}(Q(\boldsymbol{\theta}|\zeta)||P(\boldsymbol{\theta}|D))$

$$\begin{aligned}\zeta^* = \operatorname{argmin}[\mathcal{D}_{KL}[Q(\boldsymbol{\theta}|\zeta)||P(\boldsymbol{\theta}|D)]] = \operatorname{argmin} \int Q(\boldsymbol{\theta}|\zeta) \log \frac{Q(\boldsymbol{\theta}|\zeta)P(D)}{P(\boldsymbol{\theta})P(D|\boldsymbol{\theta})} d\boldsymbol{\theta} = \\ = \operatorname{argmin} \int Q(\boldsymbol{\theta}|\zeta) \left[\log \frac{Q(\boldsymbol{\theta}|\zeta)}{P(\boldsymbol{\theta})} - \log P(D|\boldsymbol{\theta}) + \log P(D)\right] d\boldsymbol{\theta} = \\ = \operatorname{argmin}[\mathcal{D}_{KL}(Q(\boldsymbol{\theta}|\zeta)||P(\boldsymbol{\theta})) - \mathbb{E}_{Q(\boldsymbol{\theta})}[\log P(D|\boldsymbol{\theta})] + \text{constant}]\end{aligned}$$

Training Phase

The minimization problem can then be summarized as finding $\zeta^* = \operatorname{argmin} \mathcal{F}(\mathcal{D}, \zeta)$, where:

$$\mathcal{F}(D,\zeta) = \mathcal{D}_{KL}(Q(\boldsymbol{\theta}|\zeta)||P(\boldsymbol{\theta})) - \mathbb{E}_{Q(\boldsymbol{\theta})}[\log P(D|\boldsymbol{\theta})] \tag{36}$$

In 36, the first term is prior-dependent and is called *complexity cost*, while the second is the classical *likelihood cost* depending on data. Our computational task will be to approximate $\mathcal{F}(D,\zeta)$ with its Monte-Carlo expectation and use it as training loss function optimized with Adam.

As auxiliary distribution Q, we choose a family that is described by a a set of parameters that we will denote by $\boldsymbol{\zeta} \in \mathbb{R}^{2d_\theta}$ following the common practice implemented also in [11], and taking a factorizable Gaussian distribution

$$Q(\boldsymbol{\theta},\boldsymbol{\zeta}) = \prod_{i=1}^{d_\theta} q(\theta_i, \zeta_i^\mu, \zeta_i^\rho) \tag{37}$$

which also allows us to compute analytically some of the quantities needed in the algorithm.

In Equation 37, the vector of parameters $\boldsymbol{\zeta}$ has as components the pairs $(\zeta_i^\mu, \zeta_i^\rho)$ for $i = 1, \dots, d_\theta$. Each pair characterizes the distribution of a network parameter, that is taken to be an uni-variate Gaussian independent one from each other and having mean ζ_i^μ and standard deviation $\log(1+\exp(\zeta_i^\rho))$. Note that we choose to work with the quantity ζ^ρ instead of the standard deviation in order to get rid of the constraint of having a positive quantity, because gradient descent-like algorithms work optimally with parameters living in the whole set $\mathbb{R}$.

Prediction Phase

The above optimization process enables us to recover the best approximating distribution in the parametric family, that we will denote as $Q(\zeta^*)$, from which we can sample as it was the real posterior distribution.

Differently with respect to algorithms like HMC, indeed, this time the process returns a distribution instead of a set of samples from a distribution, and an evident consequence of this is that we are able to generate directly an arbitrary number of samples of parameters $\{\boldsymbol{\theta}_i^*\}_i$ from the output distribution.

Again by setting each element of the set as network parameter, we can then obtain sets of samples for solution $\{\tilde{\mathbf{u}}(\boldsymbol{\theta}_i^*)\}_i$ and parametric field $\{\tilde{\mathbf{f}}(\boldsymbol{\theta}_i^*)\}_i$, to reconstruct their distribution.

3.2.2. Algorithm

We now report the complete algorithm that has been implemented:

Algorithm 5: Variational Inference

```
Initialization:
    $N$ total number of epochs
    $N_z$ number of samples of $\boldsymbol{\theta}$ to optimize $\boldsymbol{\zeta}$
    $M$ number of desired samples
    $\boldsymbol{\zeta}_i$ initial value of the auxiliary parameters
for $k = 1, \dots, N$ do
    sample $\{\mathbf{z}^{(j)}\}_{j=1}^{N_z}$ independently from $\mathcal{N}(\mathbf{0}, \mathbf{I}_{d_\theta})$
    set $\theta^{(j)} = \boldsymbol{\zeta}^\mu + \log(1+\exp(\boldsymbol{\zeta}^\rho)) \odot \mathbf{z}^{(j)} \quad j = 1, \dots, N_z$
    $L(\boldsymbol{\zeta}) = \frac{1}{N_z}\sum_{j=1}^{N_z}[\log(Q(\boldsymbol{\theta}^{(j)};\boldsymbol{\zeta})) - \log P(\boldsymbol{\theta}^{(j)}) - \log P(D|\boldsymbol{\theta}^{(j)})]$
    update $\boldsymbol{\zeta}$ with gradient $\nabla_{\boldsymbol{\zeta}} L(\boldsymbol{\zeta})$ using the Adam optimizer
sample $\{\mathbf{z}^{(j)}\}_{j=1}^{M}$ independently from $\mathcal{N}(\mathbf{0}, \mathbf{I}_{d_\theta})$
set $\boldsymbol{\theta}^{(j)} = \boldsymbol{\zeta}^\mu + \log(1+\exp(\boldsymbol{\zeta}^\rho)) \odot \mathbf{z}^{(j)} \quad j = 1, \dots, M$.
```

3.2.3. Parameters' choice

The VI method shares some characteristics with algorithms outside the Bayesian framework, as GD or Adam in the part of training of the distribution parameters; then, its final step guarantees to enter the probabilistic framework, because the output is sampled for the learned distribution.

This is reflected in the choice of the method parameters as well: the number of epochs N should be chosen having in mind the same principle as for Adam and GD, because there is an improvement of the learned parameters with the number of epochs and for the prediction only the final values of the learned parameters are used. In this stage, also the learning rate α should be chosen in a similar way to deterministic algorithms.

On the other hand, the number of samples M does not affect training and therefore computational time; with respect to methods such as SVGD, we are free to increase it to produce more significant confidence intervals, without a computational overload.

3.3. Stein Variational Gradient Descent

Another method that approximates the posterior minimizing the KL divergence with a different strategy is Stein Variational Gradient Descent (SVGD). This method can be treated as a natural counterpart of gradient descent for full Bayesian inference.

3.3.1. Theoretical Foundations

The theoretical foundations of the method are shared with VI, as the underlying major idea is still learning the posterior by minimizing the KL divergence.

Analytical Derivation

The analytical derivation of the algorithm is in this case longer than for the previous cases, hence we will simply highlight the key passages; further details can be found in the reference [8].

1. the auxiliary family Q this time consists of distributions obtained by smooth transforms T of random variable x drawn from a tractable reference distribution q_0: $z = T(x)$ with $x \sim q_0$;
2. we look for an iterative algorithm to look for the best T, which is dealt with through its perturbative formulation $T(x) = x + \varepsilon\phi(x)$, with a scheme of the form $T_{n+1}(x) = T_n(x) + \varepsilon\phi_n(x)$;
3. we insert this formulation inside the KL divergence, and since we want to minimize this quantity, we search for the steepest descent direction ϕ, that coincides with maximizing $-\nabla_\varepsilon KL(Q_T||P)|_{\varepsilon=0}$;
4. for a vectorial function $\phi(x)$, we can define the Stein operator $A_p\phi(x) = \phi(x)\nabla x \log p(x)^T + \nabla x\phi(x)$. Expanding the computations, we obtain: $-\nabla_\varepsilon KL(Q_T||P)|_{\varepsilon=0} = \mathbb{E}_{x\sim Q_T}[\mathrm{tr}(A_p(\phi(x)))]$;
5. the optimization problem now reads as maximizing the so-called kernelized stein discrepancy $\mathbb{E}_{x\sim Q_T}[\mathrm{tr}(A_p(\phi(x)))]$ in the space $RKHS$ (Reproducing Kernel Hilbert Space with kernel k);
6. the solution to this problem is known, and it is $\phi^*_{Q_T,P}(x) = \mathbb{E}_{x\sim Q_T}[k(x,\cdot)\nabla x \log(p(x)) + \nabla x k(x,\cdot)] = \mathbb{E}_{x\sim Q_T}[A_p k(x)]$;
7. finally, we want to estimate the expected value with Monte Carlo, and we can do this with N particles sampled from q_0, iteratively updated with the formula indicated in step 2.

Numerical Simulation

The intuition behind the application of this method is basically to work with a finite number of neural networks (to whom we will refer as *particles*) and transport, iteration after iteration, all the particles' parameters towards the target distribution, following the direction given by the minimization of the KL divergence.

Those particles represent samples for the computation of Monte Carlo expectation provided by Equation 38 of the derivation. At first iteration they are randomly drawn from the distribution q_0 and each time they are used to compute the Stein operator. Lastly, given the increments, we update directly the samples as they were drawn from q_1 and so on.

$$\mathbb{E}_{\boldsymbol{\theta}\sim Q}[k(\boldsymbol{\theta},\cdot)\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}} k(\boldsymbol{\theta},\cdot)], \tag{38}$$

During the analytical derivation we never mention the meaning of k; it is a strictly positive definite kernel and one suitable example is the RBF kernel that we exploit in our implementation:

$$k(x,x') = \exp\left(-\frac{1}{h}||x-x'||^2\right) \tag{39}$$

with $h > 0$ positive bandwidth.

Note that to the two addends in Equation 38 we can associate a specific meaning:

1. $k(\boldsymbol{\theta},\cdot)\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta})$ drives the particles towards the high probability areas of p (or of $\log p$, that is $L(\boldsymbol{\theta})$ of Equation 26) by following a smoothed gradient direction, which is the weighted sum of the gradients of all the points weighted by the kernel function;
2. $\nabla_{\boldsymbol{\theta}} k(\boldsymbol{\theta},\cdot)$ acts as a *repulsive force* that prevents all the points to collapse together into the same point, that is the maximum-a-posteriori of $L(\boldsymbol{\theta})$. Indeed, considering the RBF kernel, we have that:

$$\nabla_x k(x,x') = \frac{2}{h}(x-x')k(x,x') \tag{40}$$

   hence x is driven away from neighbours that are such that $k(x,x')$ is big. If we let the bandwidth $h \to 0$, the repulsive term vanishes: this means that the parameters' update reduces to a set of independent chains of typical gradient ascent for maximizing $\log p$ and all the particles would collapse into the local modes.

The update of the parameters can then be done as in gradient descent introducing a term that plays the role of the *learning rate* as well. Moreover, we could also implement an approach similar to *stochastic gradient descent* (algorithm 2), that can be useful when dealing with large datasets.

3.3.2. Algorithm

We now report the complete algorithm that has been implemented:

Algorithm 6: Stein Variational Gradient Descent

```
Initialization:
        $N$ number of neural networks
        $\{\boldsymbol{\theta}_i\}_{i=1}^N$ initial value of parameters for each network
        $E$ total number of epochs
        $\varepsilon$ step size
for $k = 1,\dots,E$ do
    $\phi(\boldsymbol{\theta}_i) = \frac{1}{N}\sum_{j=1}^N k(\boldsymbol{\theta}_i,\boldsymbol{\theta}_j)\nabla_{\boldsymbol{\theta}_j} L(\boldsymbol{\theta}_j) + \nabla_{\boldsymbol{\theta}_j} k(\boldsymbol{\theta}_i,\boldsymbol{\theta}_j) \quad \forall i = 1,\dots,N$
    $\boldsymbol{\theta}_i = \boldsymbol{\theta}_i + \varepsilon\phi(\boldsymbol{\theta}_i) \quad \forall i = 1,\dots,N$
collect the parameters of all networks after $E$ epochs $\{\boldsymbol{\theta}^{(i)}\}_{i=1}^N$.
```

3.3.3. Parameters' choice

In terms of parameters selection, the decision effort required by SVGD is not that high, especially compared to HMC. In fact, the parameters to be set are only N, E and ε, but the real obstacle for tuning consists in the fact that SVGD is the most computational demanding algorithm, issue that could be partially solved with a parallelization of the computations made by the particles.

The number of epochs E affects the quality of the prediction, while the number of particles N can determine the quality of the UQ; in [8] it is suggested that the convergence to the target distribution is reached for $E, N \to \infty$, hence (clearly) the bigger the E and the N, the better the approximation of the distribution. On the other hand, large E and N require higher computational time, and moreover enough RAM memory to store the parameters of all the particles.

The learning rate ε played a fundamental role during parameters' tuning, as learning rates parameters for other common algorithms, because its choice enabled to prevent the network's trainable parameters to explode to `nan`, due to a growth out of control.