Stein Variational Gradient Descent

Omar Hagrass

- MCMC is not very suitable for very high dimensions (Slow , not easy to converge in practice ,hard to choose q(x))

- Variational methods frames this problem from another perspective by turning it into a deterministic optimization that minimizes the KL divergence between target distribution and a set of simpler distributions

- The **main goal** is to draw samples from some complicated distribution and use it to approximate E(G(x))

Kullback–Leibler divergence

$$KL(q||p) = \int \log\left(\frac{q(x)}{p(x)}\right) q(x)dx$$

Valid distance >= 0
q(x)=p(x) <=> 0

The Goal

$$q^* = \arg\min_{q\in\mathcal{Q}} \{\mathrm{KL}(q \,||\, p) \equiv \mathbb{E}_q[\log q(x)] - \mathbb{E}_q[\log \bar{p}(x)] + \log Z\}$$

Up to a normalization constant is enough

The set of distribution should satisfy :
-accuracy(broad enough)
-solvability (the optimization problem can be solved efficiently)

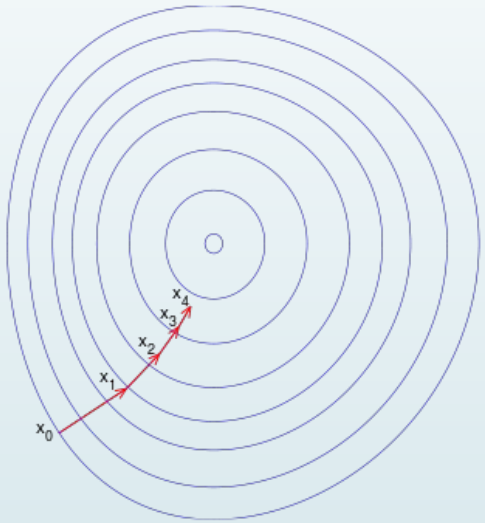
Here it is chosen to be the set of smooth transformations:
Let $z = T(x),\ then\ q_T(z) = q\left(T^{-1}(z)\right).|\det(\nabla_z T^{-1}(z))|$
-This is broad enough (Villani 2008 , optimal transform)
- Solvability (needs some assumptions later)

$$KL(q_T||p) = \int \log\left(\frac{q\left(T^{-1}(x)\right).|\det(\nabla_x T^{-1}(x))|}{p(x)}\right) q\left(T^{-1}(x)\right).|\det(\nabla_x T^{-1}(x))|dx$$

Which T(x) that minimizes this distance ? No analytical solution , need iterative algorithm

Gradient descent

- Start at some x
- x_new= x + h (h is the perturbation direction)
- Now what is best h to minimizes f(x) most at this step?
- $\frac{df(x+h)}{dh}|_{h=0} = f'(x)$ i.e the gradient at current x
- so if we choose $h = \delta f'(x)$, the f(x) decreases by this amount



Let $T(x) = x + \epsilon\phi(x)$, a small perturbation then this will give

$$\nabla_\epsilon \mathrm{KL}(q_{[T]} \,||\, p)\,\big|_{\epsilon=0} = -\mathbb{E}_{x\sim q}[\mathrm{trace}(\mathcal{A}_p\phi(x))],$$

$$\mathcal{A}_p\phi(x) = \nabla_x \log p(x)\phi(x)^\top + \nabla_x\phi(x) \textit{ is the Stein operator.}$$

Thus , choose $\phi(x)^*$ that maximizes the expectation ,this is the steepest descent direction

- $\phi^*(x) = \text{argmax}_{\phi \in H^d} E_q(\, trace(A_p(\phi(x)))$

- It turns out if we choose H^d to be RKHS (reproducing kernel Hilbert space) then there is closed form solution

$$\phi^*_{q,p}(\cdot) = \mathbb{E}_{x\sim q}[\mathcal{A}_p k(x,\cdot)]$$

$$\phi^*_{q,p}(\cdot) = \mathbb{E}_{x\sim q}[k(x,\cdot)\nabla_x \log p(x) + \nabla_x k(x,\cdot)],$$

RKHS: $\{f : f(x) = \sum_{i=1}^{m} a_i k(x, x_i),\quad a_i \in \mathbb{R},\ m \in \mathbb{N},\ x_i \in \mathcal{X}\}.$

- Where k(x,y) is the reproducing kernel , example : k(x,y)=exp$(-\frac{1}{h}||x-y||^2)$

Note on stein operator: $\mathcal{A}_p\phi(x) = \nabla_x \log p(x)\phi(x)^\top + \nabla_x \phi(x)$ *is the Stein operator.*

- $\mathbb{E}_{x\sim p}[\mathcal{A}_p\phi(x)] = 0$

So if q -> p the gradient should go to zero

- Stein discrepancy : $S(p||q) = \max_{\phi \in H^d} E_q(\, trace(A_p(\phi(x)))$

The Algorithm:

$$x_i^{\ell+1} \leftarrow x_i^{\ell} + \epsilon_\ell \hat{\phi}^*(x_i^{\ell}) \quad \text{where} \quad \hat{\phi}^*(x) = \frac{1}{n}\sum_{j=1}^{n} \left[k(x_j^{\ell}, x)\nabla_{x_j^{\ell}} \log p(x_j^{\ell}) + \nabla_{x_j^{\ell}} k(x_j^{\ell}, x)\right].$$

Intuition :

-The first term drives the particles towards the high probability areas of p(x) by following a smoothed gradient direction, which is the weighted sum of the gradients of all the
points weighted by the kernel function.

-The second term acts as a repulsive force that prevents
all the points to collapse together into local modes of p(x)
Consider k(x,y)=$\exp(-\frac{1}{h}||x-y||^2)$ then second term = $\sum_{j=1}^{n}(x - x_j)k(x_j, x)$

- Single particle case
- Not iid samples but better for approximation , **completely deterministic no randomness**

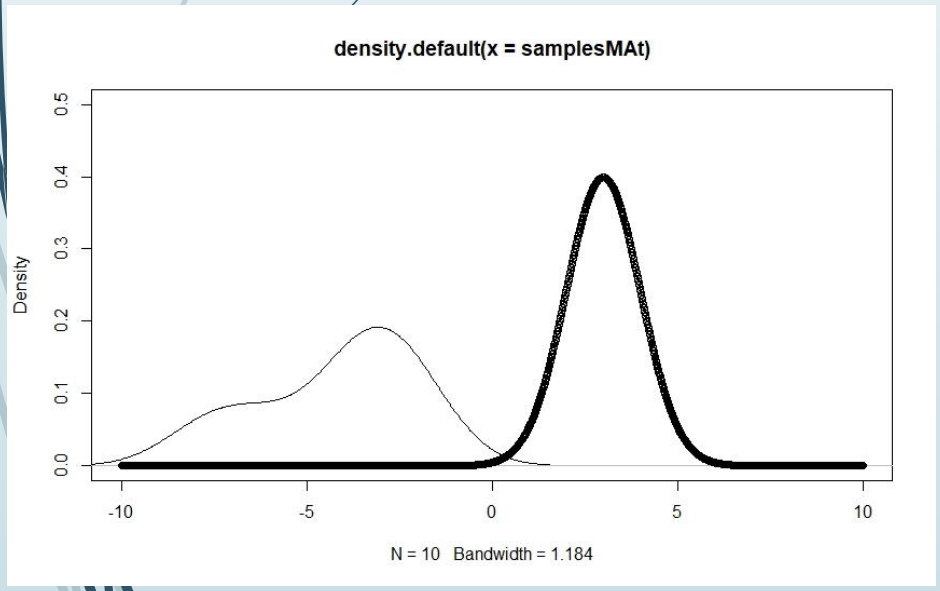
Computational properties

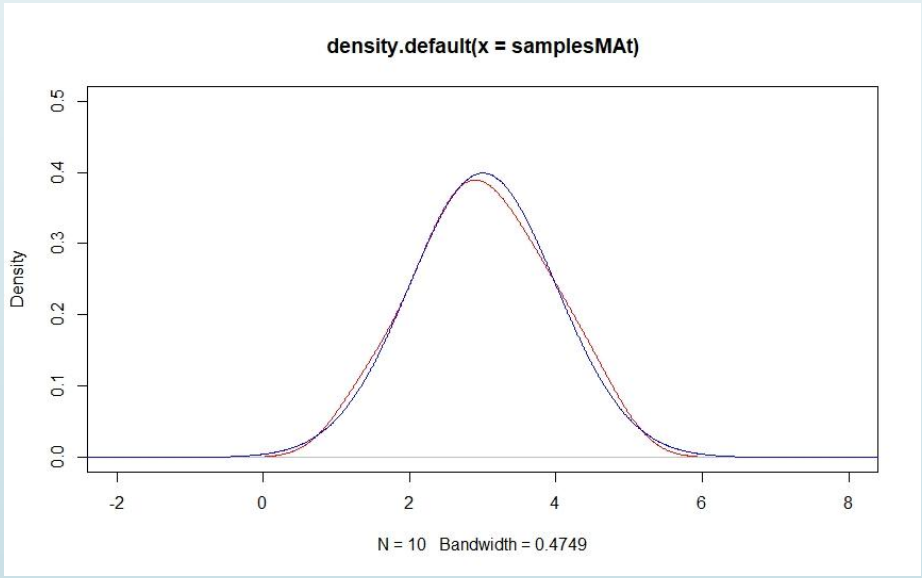
$$x_i^{\ell+1} \leftarrow x_i^{\ell} + \epsilon_\ell \hat{\phi}^*(x_i^{\ell}) \quad \text{where} \quad \hat{\phi}^*(x) = \frac{1}{n}\sum_{j=1}^{n} \left[k(x_j^{\ell}, x)\nabla_{x_j^{\ell}} \log p(x_j^{\ell}) + \nabla_{x_j^{\ell}} k(x_j^{\ell}, x)\right].$$

- Don't need to be smart with the choice of q(x) , you can start from any random particles.
- Bottleneck is to compute $\nabla_x \log\big(p(x)\big)$ for all points. So if is formed from a data points we can use subsamples of the data.
- In practice small n is enough for approximations (around 200)
- I used finite difference to approximate derivatives so the code will work generally for any p(x)
- The algorithm can parallelized in each iteration
- It can be written more efficiently in matrix form

Simulations

- Baby Toy example , p(x) = N(3,1) , n=10 , after 1000 iteration ,
- $\mathrm{E}(x) = 2.996 , E(x^2) = 9.74$



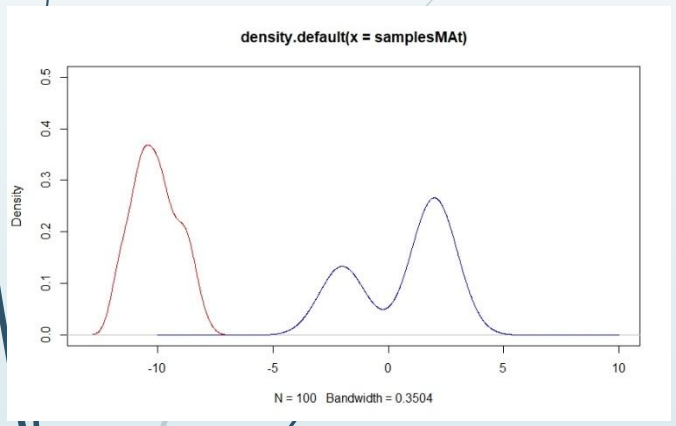


p(x) Blue

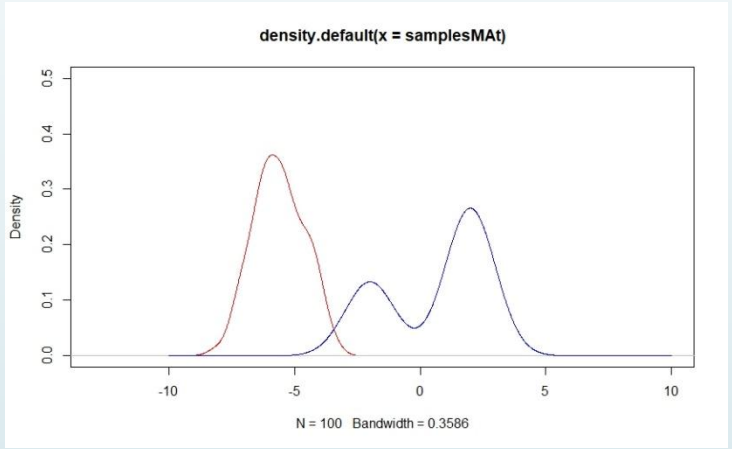
Toy example : $p(x) = \left(\frac{1}{2}\right) N(-2,1) + \left(\frac{2}{2}\right) N(2,1)$

N= 100

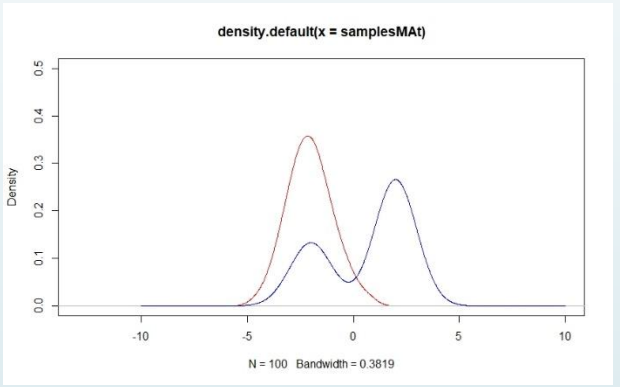
0 iter



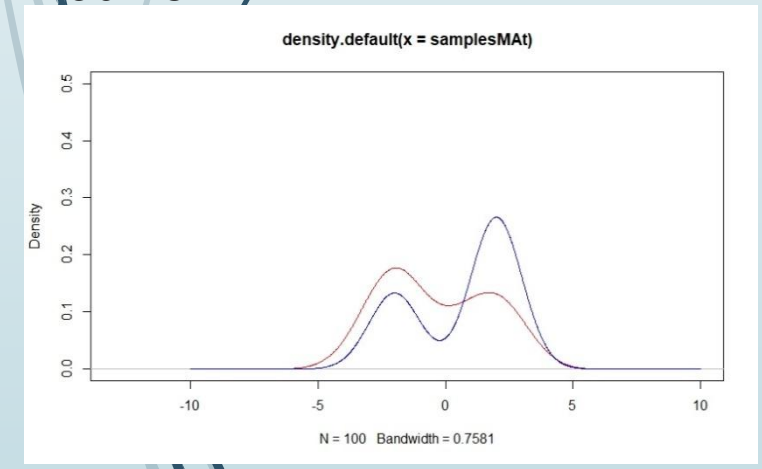
100 iter



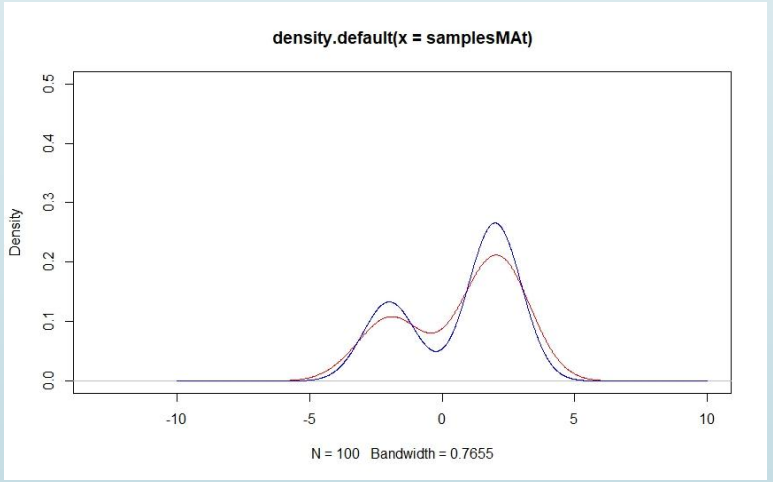
200 iter



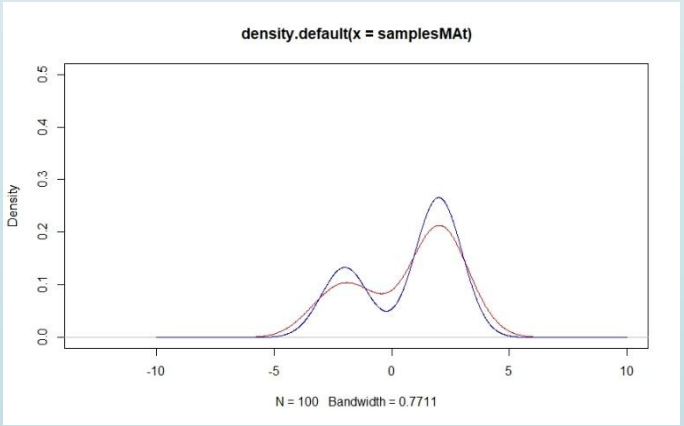
250 iter



350 iter

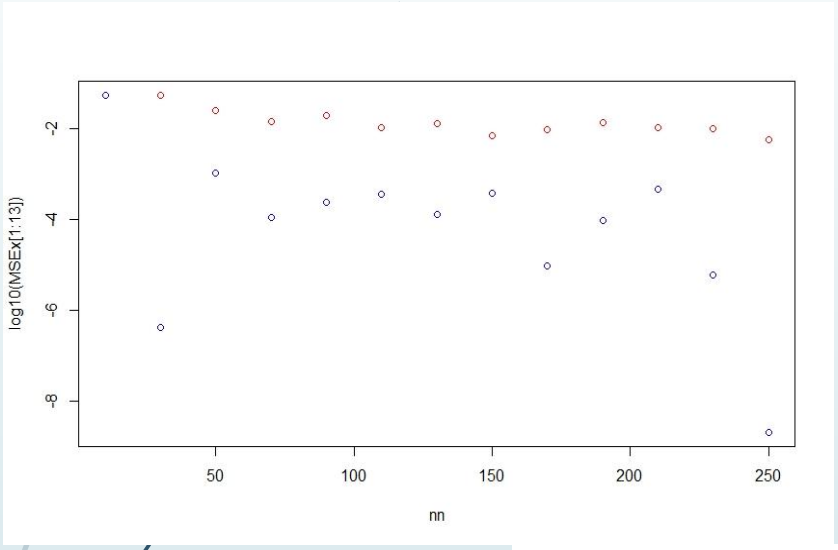


1000 iter

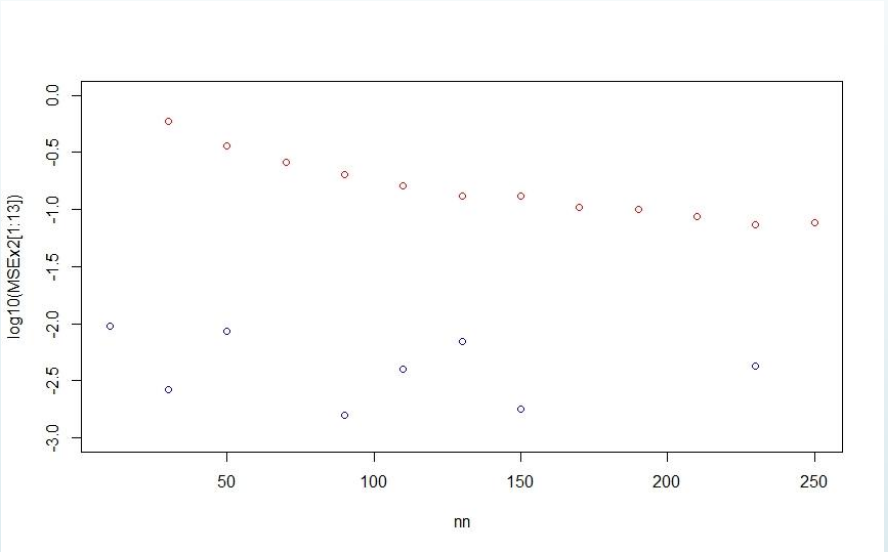


Comparison with monte Carlo samples in terms of MSE

E(x)



E(x^2)



E(cos(0.5x+1))