

# Boosted Variational Inference via Bayesian Monte Carlo

**Danilo de Freitas Naiff**

**IM-UFRJ**

Sept. 23nd 2019

- 1 Introduction
- 2 Gaussian Processes
- 3 Bayesian Monte Carlo
- 4 Variational Inference
- 5 Boosted Variational Bayesian Monte Carlo
- 6 Experiments
- 7 Challenges and conclusion

## Bayesian theory

Objective: update knowledge of parameter  $\theta$  given data  $\mathcal{D}$ .

Building blocks

- Prior probability  $p(\theta)$
- Likelihood  $p(\mathcal{D}|\theta)$

Posterior probability

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{\int p(\mathcal{D}|\theta')p(\theta')d\theta'}$$

Using posterior probability:

$$\langle f(\theta) \rangle = \int_{\Theta} f(\theta)p(\theta|\mathcal{D})d\theta$$

## Bayesian theory

Objective: update knowledge of parameter  $\theta$  given data  $\mathcal{D}$ .

Building blocks

- Prior probability  $p(\theta)$
- Likelihood  $p(\mathcal{D}|\theta)$

Posterior probability

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{\int p(\mathcal{D}|\theta')p(\theta')d\theta'}$$

Using posterior probability:

$$\langle f(\theta) \rangle = \int_{\Theta} f(\theta)p(\theta|\mathcal{D})d\theta$$

## Bayesian theory

Objective: update knowledge of parameter  $\theta$  given data  $\mathcal{D}$ .

Building blocks

- Prior probability  $p(\theta)$
- Likelihood  $p(\mathcal{D}|\theta)$

Posterior probability

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{\int p(\mathcal{D}|\theta')p(\theta')d\theta'}$$

Using posterior probability:

$$\langle f(\theta) \rangle = \int_{\Theta} f(\theta)p(\theta|\mathcal{D})d\theta$$

## Bayesian theory

Objective: update knowledge of parameter  $\theta$  given data  $\mathcal{D}$ .

Building blocks

- Prior probability  $p(\theta)$
- Likelihood  $p(\mathcal{D}|\theta)$

Posterior probability

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{\int p(\mathcal{D}|\theta')p(\theta')d\theta'}$$

Using posterior probability:

$$\langle f(\theta) \rangle = \int_{\Theta} f(\theta)p(\theta|\mathcal{D})d\theta$$

## Approximate inference

Ways to integrate:

- Monte Carlo

$$\int_{\Theta} f(\theta) p(\theta|\mathcal{D}) d\theta \approx \frac{1}{N} \sum_{i=1}^N f(\theta_i), \quad \theta_i \sim p(\theta|\mathcal{D})$$

- Approximate distribution

$$p(\theta|\mathcal{D}) \approx q(\theta), \quad \langle f(\theta) \rangle \approx \int_{\Theta} f(\theta) q(\theta) d\theta$$

Usual methods demands many evaluations of  $p(\mathcal{D}|\theta)p(\theta)$ . However, *this is not always feasible.*

## Approximate inference

Ways to integrate:

- Monte Carlo

$$\int_{\Theta} f(\theta) p(\theta|\mathcal{D}) d\theta \approx \frac{1}{N} \sum_{i=1}^N f(\theta_i), \quad \theta_i \sim p(\theta|\mathcal{D})$$

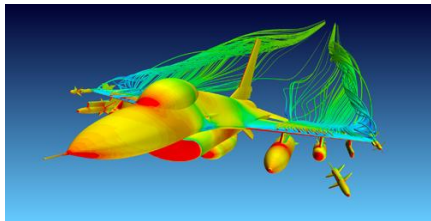
- Approximate distribution

$$p(\theta|\mathcal{D}) \approx q(\theta), \quad \langle f(\theta) \rangle \approx \int_{\Theta} f(\theta) q(\theta) d\theta$$

Usual methods demands many evaluations of  $p(\mathcal{D}|\theta)p(\theta)$ . However,  
*this is not always feasible.*



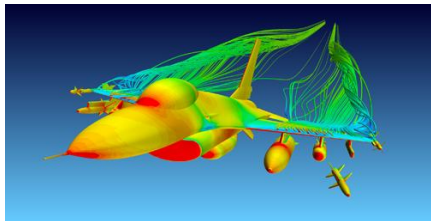
In science, there are many cases that  $p(\mathcal{D}|\theta)$  demands the computation of a forward model  $g(\theta)$ , which comes from an expensive simulation.



This requires approximate inference methods "on a budget". In this work, one such method is developed, based on preexisting work. We name it

*Boosted Variational Bayesian Monte Carlo (BVBMC).*

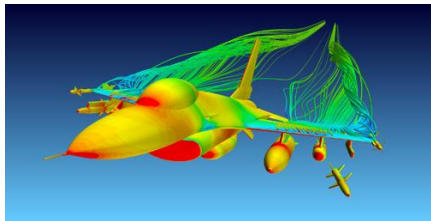
In science, there are many cases that  $p(\mathcal{D}|\theta)$  demands the computation of a forward model  $g(\theta)$ , which comes from an expensive simulation.



This requires approximate inference methods "on a budget". In this work, one such method is developed, based on preexisting work. We name it

*Boosted Variational Bayesian Monte Carlo (BVBMC).*

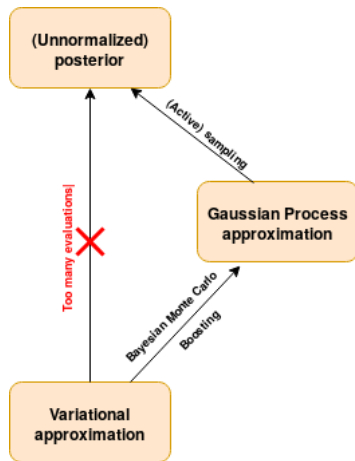
In science, there are many cases that  $p(\mathcal{D}|\theta)$  demands the computation of a forward model  $g(\theta)$ , which comes from an expensive simulation.



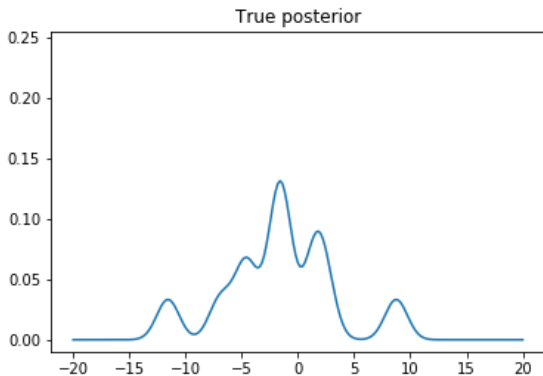
This requires approximate inference methods "on a budget". In this work, one such method is developed, based on preexisting work. We name it

*Boosted Variational Bayesian Monte Carlo (BVBMC).*

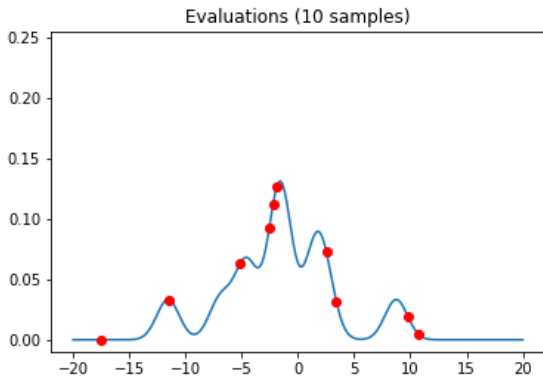
## BVBMC schema



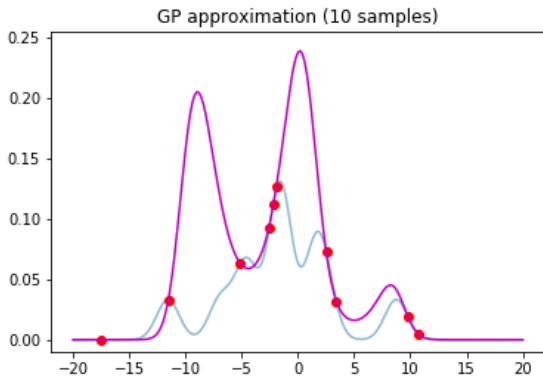
## An illustration of BVBMC



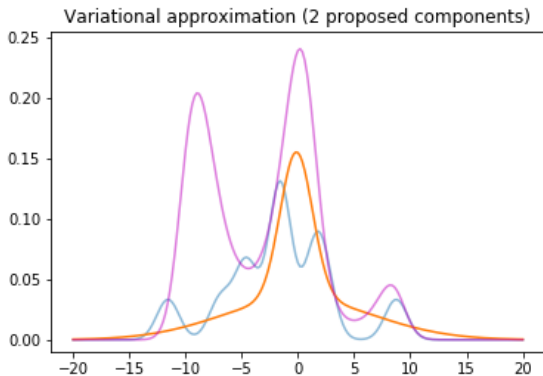
## An illustration of BVBMCMC



## An illustration of BVBMCM

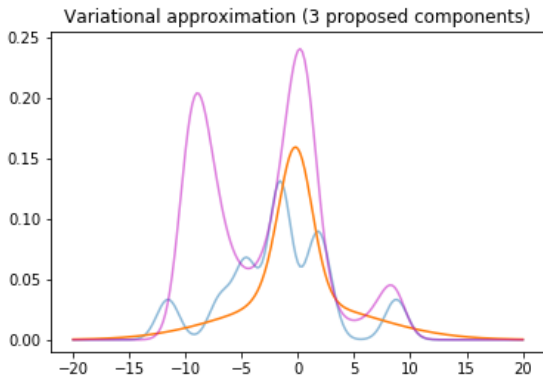


## An illustration of BVBMC

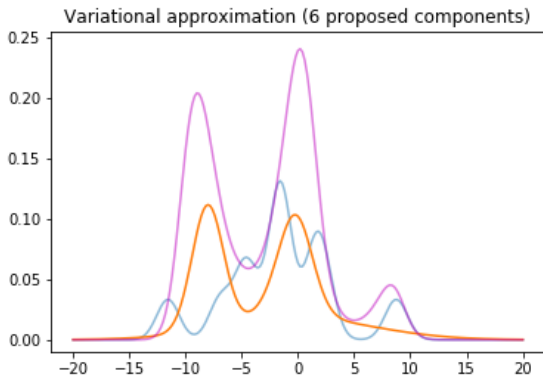




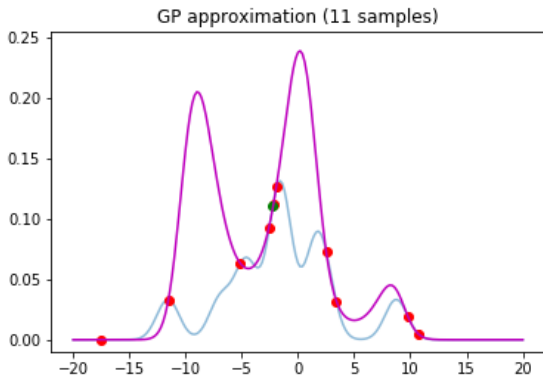
## An illustration of BVBMC



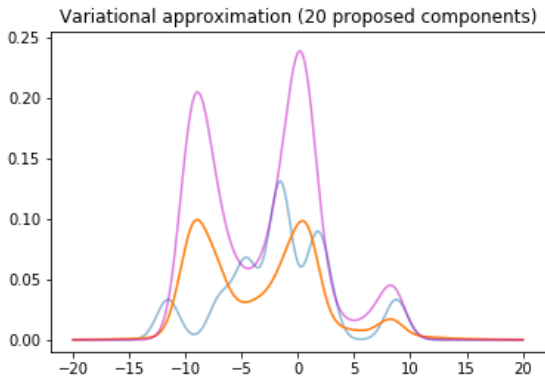
## An illustration of BVBMC



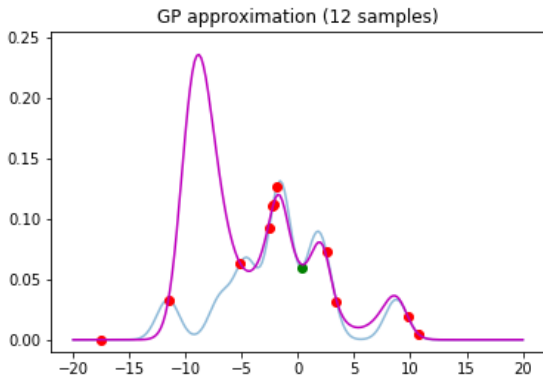
## An illustration of BVBMCMC



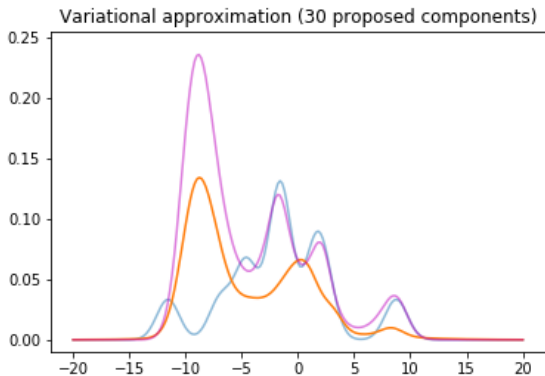
## An illustration of BVBM



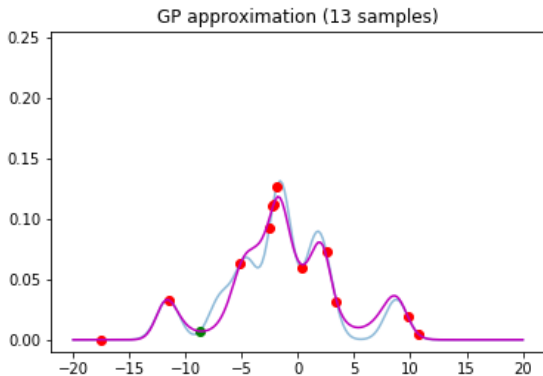
## An illustration of BVBMCM



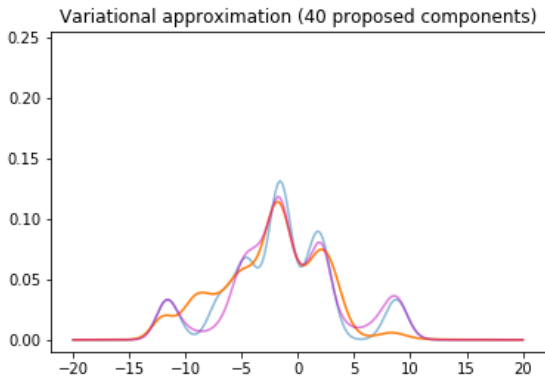
## An illustration of BVBMC



## An illustration of BVBMC

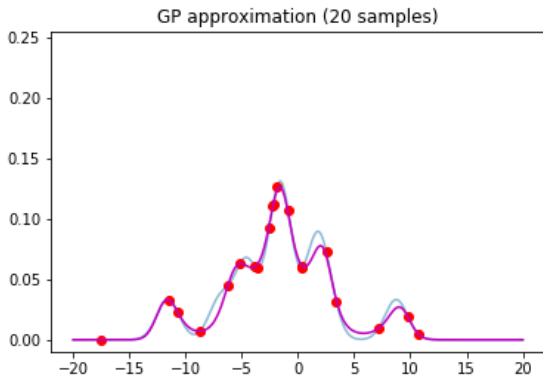


## An illustration of BVBMC

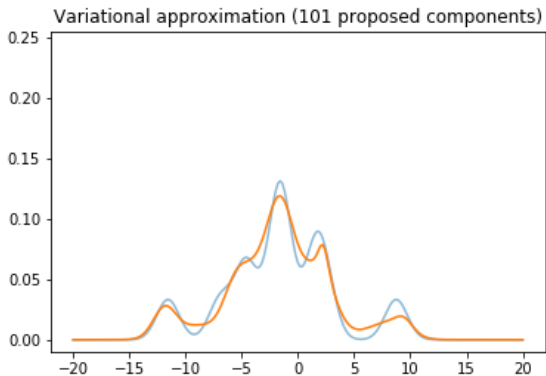




## An illustration of BVBMCM



## An illustration of BVBMC



## Definition

Gaussian processes (GP): distribution over functions  $f : \mathcal{X} \rightarrow \mathbb{R}$  such that  $f(\mathbf{x}) = (f(x_1), \dots, f(x_n))$  follows a multivariate normal distribution. A GP is completely defined by:

- $m(x; \theta) := \mathbb{E}[f(x)]$ , mean function.
- $k(x, x'; \theta) := \text{Cov}[f(x), f(x')]$ , covariance function or kernel.

such that  $f(\mathbf{x}) \sim \mathcal{N}(m(\mathbf{x}), K(\mathbf{x}, \mathbf{x}))$ .

## Gaussian process regression

Given  $\mathcal{D} = (x, y)_{i=1}^N$ , a Gaussian process regression is made by assuming  $p(y|x) = p(y|f(x))$ , with  $f$  following a prior  $GP(m, k)$ .

## Posterior GP

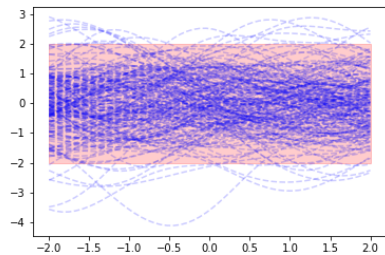
If  $p(y|f(x)) = \mathcal{N}(f(x), \sigma_n^2)$ ,  $f|\mathcal{D} \sim GP(m_{\mathcal{D}}, k_{\mathcal{D}})$ , where

$$m_{\mathcal{D}}(x) := m(x) + K(x^*, \mathbf{x})(K(\mathbf{x}, \mathbf{x}) + \sigma_n^2 I)^{-1}(\mathbf{y} - m(\mathbf{x}))$$

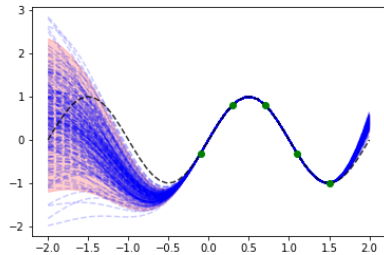
$$k_{\mathcal{D}}(x, x') := k(x, x') - K(x, \mathbf{x})(K(\mathbf{x}, \mathbf{x}) + \sigma_n^2 I)^{-1}K(\mathbf{x}, x')$$

Reduces to deterministic measurement when  $\sigma_n^2 = 0$ . More general  $p(y|f(x))$  must resort to explicit marginalization.

## Example case



GP prior



GP posterior

## Kernels

The assumption that  $K(\mathbf{x}, \mathbf{x})$  is a covariance matrix restricts which functions can be kernels. Some examples of kernels in  $\mathbb{R}$  are:

- $k_{SQE}(x, x'; \theta_0, l) = \theta_0 \exp\left(-\frac{1}{2} \frac{(x-x')^2}{l^2}\right)$
- $k_{\text{Matern}, 3/2}(x, x'; \theta_0, l) = \theta_0 \left(\sqrt{3} \frac{(x-x')}{l}\right) \exp\left(-\sqrt{3} \frac{(x-x')}{l}\right)$

Kernels in  $\mathbb{R}^D$  can be constructed by changing  $\frac{(x-x')}{l}$  for  $\sqrt{\sum_{i=1}^D \frac{(x_i-x'_i)^2}{l_i^2}}$ .

If  $k_1, k_2$  are kernels, the following, among others are kernels:

$$k_1(x, x') + k_2(x, x'), k_1(x, x')k_2(x, x'), k_1(x, x')k_2(y, y'), k_1(f(y), f(y')).$$

## Mean functions

In general, they are less important than kernels, since the latter determines the structure of the posterior GP. However, *outside the sampling area the GP prediction defaults to the mean*, which may be of importance.

## Handling hyperparameters

$$\log p(\mathcal{D}|\theta) = -\frac{1}{2}(\mathbf{y} - m(\mathbf{x}))^T (K(\mathbf{x}, \mathbf{x}) + \sigma_n \mathbf{I})^{-1} (\mathbf{y} - m(\mathbf{x})) + \\ -\frac{1}{2} \log \det(K(\mathbf{x}, \mathbf{x}) + \sigma_n \mathbf{I}) - \frac{1}{2} N \log(2\pi).$$

Inference can be done either by MLE, MAP, or integration techniques.

## Scaling

The bottleneck of GP regression:  $(K(\mathbf{x}, \mathbf{x}) + \sigma_n \mathbf{I})^{-1}$ . Cost is  $\mathcal{O}(N^3)$ .  
In online learning, each new sample is incorporated in  $\mathcal{O}(N^2)$ .

## Integrating a GP

As discussed, often one wants to take expectations

$$Z = \int f(x)p(x)dx$$

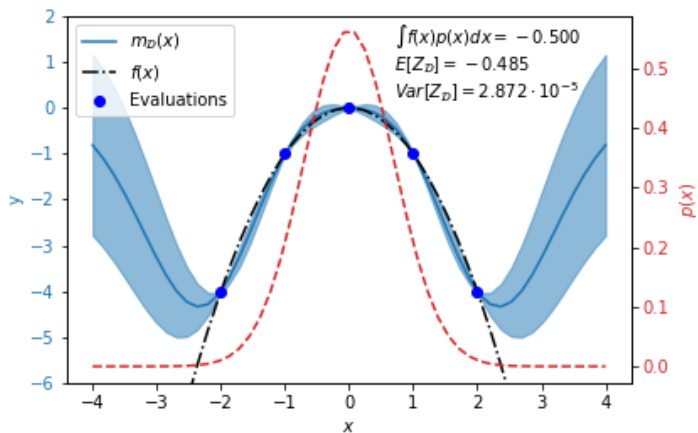
Bayesian Monte Carlo: given  $\mathcal{D} = \{x_i, f(x_i)\}_{i=1}^N$ , approximate  $f$  by  $f_{\mathcal{D}} \sim GP(m_{\mathcal{D}}, k_{\mathcal{D}})$ . This makes

$$Z_{\mathcal{D}} = \int f_{\mathcal{D}}(x)p(x)dx$$

be a Gaussian random variable

Name Bayesian *Monte Carlo* is misleading.





## Mean and variance for BMC

$$\mathbb{E}[Z_D] = \int m(x)p(x)dx - \mathbf{z}^T K^{-1}(\mathbf{f} - m(\mathbf{x})), \quad \text{Var}[Z_D] = \Gamma - \mathbf{z}^T K^{-1} \mathbf{z},$$

$$z_i = \int k(x, x_i)p(x)dx, \quad \Gamma = \int \int k(x, x')p(x)p(x')dxdx'.$$

## Kernel integral terms

In the general case, they can be estimated by Monte Carlo. When  $p(x)$  is Gaussian or a mixture of Gaussians:

- Analytically tractable when  $k(x, x')$  is the SQE kernel.
- Efficiently tractable when  $k(x, x') = k(x_1, y_1) \dots k(x_D, y_D)$ .

## Active evaluation

Given  $\{(x_1, f(x_1)), \dots, (x_N, f(x_N))\}$ ,  $x_{N+1}$  may be chosen by maximizing *acquisition functions*.

$$\alpha^N(x) = \alpha(x; \{(x_1, f(x_1)), \dots, (x_N, f(x_N))\})$$

Examples:

- For general integrands

$$\alpha_{\text{US}}^N(x) = k_{\mathcal{D}}(x, x)p(x)^2$$

- For positive integrands

$$\alpha_{\text{MMLT}}^N(x) = e^{2m_{\mathcal{D}}(x) + k_{\mathcal{D}}(x, x)} \left( e^{k_{\mathcal{D}}(x, x)} - 1 \right).$$

## Back to approximating posteriors

$$p(\theta|\mathcal{D}) = g(\theta) \approx q(\theta; \lambda)$$

Choosing  $q$ : minimizing some measure of dissimilarity. The family  $q(\theta; \lambda)$  must be easy to treat, in order for the approximation to be useful.

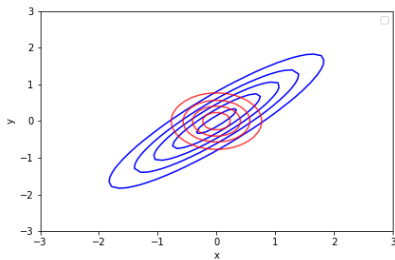
Variational inference: uses  $D_{KL}(q(\cdot; \lambda)||g)$  for measure of dissimilarity

$$D_{KL}(q||g) = - \int \log \frac{p(\theta)}{q(\theta)} q(\theta) d\theta$$

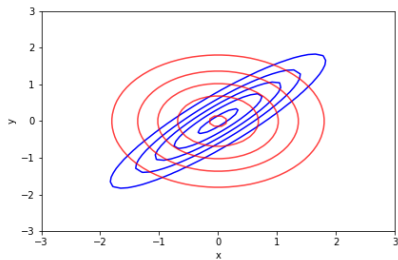
### $D_{KL}(q||g)$ vs $D_{KL}(g||q)$

$D_{KL}(q||g) \geq 0$ ,  $D_{KL}(q||g) = 0 \iff q = g$ , but  $D_{KL}(q||g) \neq D_{KL}(g||q)$ . Then, KL divergence minimization gives two different algorithms, the second one being *expectation propagation*.

## Illustration



$$D_{KL}(q||g)$$



$$D_{KL}(g||q)$$

## Evidence lower bound

Usually, one only has access to  $p(\mathcal{D}|\theta)p(\theta) = \bar{g}(\theta)$ . Fortunately, minimizing  $D_{KL}(q||g)$  is equivalent to maximizing the *evidence lower bound* (ELBO).

$$\mathcal{L}_{\bar{g}}(q) = \int \log \bar{g}(\theta) q(\theta) d\theta - \int \log q(\theta) q(\theta) d\theta$$

Moreover,  $\mathcal{L}_{\bar{g}}(q) \leq \log p(\mathcal{D})$ . Can be used for model selection.

## Maximizing the ELBO

For parameterized  $q(\theta; \lambda)$ : access to stochastic estimation of  $\nabla \mathcal{L}(\lambda)$  can be used for stochastic gradient ascent.

Reparametrization: if samples  $X_\lambda \sim q(\theta; \lambda)$  can be written as  $s(Y, \lambda)$ , with  $Y \sim r(\epsilon)$ :

$$\nabla \mathcal{L}(\lambda) = \nabla \left( \mathbb{E}_{r(\epsilon)} \left[ \log \frac{\bar{g}(s(\epsilon; \lambda))}{q(s(\epsilon; \lambda); \lambda)} \right] \right) \approx \frac{1}{K} \sum_{Y_i \sim r(\epsilon)} \nabla \left( \log \frac{\bar{g}(s(Y_i; \lambda))}{q(s(Y_i; \lambda); \lambda)} \right).$$

## Mixture of Gaussians

$q_k(\theta; \lambda) = \sum_{i=1}^k w_i f_i(\theta) = \sum_{i=1}^k w_i \mathcal{N}(\theta; \mu, \Sigma_i)$ . Analytical mean and covariance. Samples can be easily generated. Are in a sense universal approximators. Good choice for variational approximation.

## Parameterizing mixtures of Gaussian

Covariance parameterizations:

- $\Sigma_i = \text{diag}(\sigma_{i,1}^2, \dots, \sigma_{i,D}^2)$
- $\Sigma_i = \mathbf{u}_i \mathbf{u}_i^T + \text{diag}(\sigma_{i,1}^2, \dots, \sigma_{i,D}^2)$

Weights parameterizations  $w_i(\nu_i) = \frac{\phi(\nu_i)}{\sum_{k=1}^k \phi(\nu_k)}$ .  $\phi(\nu)$  can be  $\exp(\nu)$  or  $\text{softplus}(\nu) = \log(1 + \exp(\nu))$

## ELBO for mixtures of Gaussians

$$\mathcal{L}(\lambda) = \sum_{i=1}^k w_i(\nu_i) \mathbb{E}_{\epsilon \sim \mathcal{N}(0, I)} \left[ \log \frac{\bar{g}(s(\epsilon; \mu_i, \sigma_i))}{q_k(s(\epsilon; \mu_i, \sigma_i); \lambda)} \right]$$

## Boosting mixtures

Problem: no way to know how many mixtures is needed. Adding mixtures sequentially can become costly. One solution: boosting.

$$q_{i-1}(\theta) = \sum_{j=1}^{i-1} w_j f_j(\theta)$$

$$q_i(\theta) = \sum_{j=1}^{i-1} (1 - w_i) w_j f_j(\theta) + w_i f_i(\theta)$$

How to find  $w_i$  and  $f_i(\theta) = \mathcal{N}(\theta; \mu_i, \Sigma_i)$ ?

## Setting component weights

Given component  $f_i(\theta)$ , maximize  $\mathcal{L}_i(w_i) = \mathcal{L}((1 - w_i)q_{i-1}(\theta) + w_i f_i(\theta))$  via its derivative

$$\begin{aligned} \mathcal{L}'_i(w_i) = & \int \log(\bar{g}(\theta))(f_i(\theta) - q_{i-1}(\theta))d\theta - \\ & \int \log((1 - w_i)q_{i-1}(\theta) + w_i f_i(\theta))(f_i(\theta) - q_{i-1}(\theta))d\theta. \end{aligned}$$

Fortunately,  $\mathcal{L}_i(w_i)$  is a concave objective.



## Finding components

Gradient boosting: technique for finding new components.

$$f_i = \arg \min_f \nabla D_{KL}(q_{i-1} || g) \cdot f = \arg \min_f \int \log \frac{q_{i-1}(\theta)}{g(\theta)} f(\theta) d\theta.$$

Problem: degenerate solution. Needs regularization.

Maximization objective for mixture of Gaussians:

$$\begin{aligned} \text{RELBO}(\mu_i, \Sigma_i) = & \int \log(\bar{g}(\theta)) \mathcal{N}(\theta | \mu_i, \Sigma_i) d\theta - \\ & \int \log(q_{i-1}(\theta)) \mathcal{N}(\theta | \mu_i, \Sigma_i) d\theta + \frac{\lambda}{4} \log |\Sigma|, \end{aligned}$$

Reparameterization trick can be used.

```

1: procedure VARIATIONALBOOSTING( $\log \bar{g}, \mu_0, \Sigma_0$ )
2:    $\triangleright \mu_0, \Sigma_0$  the are initial boosting values
3:    $w_0 := 1.0$ 
4:   for  $t = 1, \dots, T$  do
5:      $\mu_t, \Sigma_t := \operatorname{argmax} RELBO(\mu_t, \Sigma_t)$   $\triangleright$  Using reparameterization
6:      $w_t := \operatorname{argmax} \mathcal{L}_i(w_i)$   $\triangleright$  Using  $\mathcal{L}'_t(w_t)$  for gradient descent
7:     for  $j = 0, \dots, t - 1$  do
8:        $w_j \leftarrow (1 - w_t)w_j$ 
9:     end for
10:  end for
11:  return  $\{(\mu_t, \Sigma_t, w_t)\}_{t=1}^T$ 
12: end procedure

```

## Variational Bayesian Monte Carlo (VBMC)

$$\mathcal{L}(\lambda) = \int \log \bar{g}(\theta) q(\theta; \lambda) d\theta - \int \log q(\theta; \lambda) q(\theta; \lambda) d\theta$$

Use Bayesian Monte Carlo:

$$\mathcal{L}_{\mathcal{D}}(\lambda) = \int \log \bar{g}_{\mathcal{D}}(\theta) q(\theta; \lambda) d\theta - \int \log q(\theta; \lambda) q(\theta; \lambda) d\theta$$

$$\text{Maximize } \mathbb{E}[\mathcal{L}_{\mathcal{D}}(\lambda)] = M(\lambda) + \mathbf{z}^T \mathbf{w} - \int \log q(\theta; \lambda) q(\theta; \lambda) d\theta$$

$$\mathbf{w} = K^{-1} \mathbf{y}$$

$$M(\lambda) = \int m(\theta) q(\theta; \lambda) d\theta$$

$$\mathbf{z}_i = \int k(x, x_i) q(\theta; \lambda) dx.$$

## Mean function

$m(\theta) = 0$ :  $\log \bar{g}_D(\theta)$  is not a log probability

Principled solution:  $m(\theta) = -\frac{1}{2} \sum_{i=1}^D \frac{(\theta_i - c_i)^2}{\ell_i^2}$ . Lends analytical  $M(\lambda)$ .

Ad-hoc solution:  $m(\theta) = C$ , with  $C$  being a large negative constant.

## Active evaluation

Just as in BMC, it is possible to do active evaluation. Some options:

- $\alpha_{\text{US}}^D(\theta_{N+1}) = k_D(\theta_{N+1}, \theta_{N+1}) q_k(\theta_{N+1}; \lambda)^2$ .
- $\alpha_{\text{PROP}}^D(\theta_{N+1}) = k_D(\theta_{N+1}, \theta_{N+1}) \exp(m_D(\theta_{N+1})) q_k(\theta_{N+1}; \lambda)^2$

## BVBMC

BVBMC = VBMC + boosting + small changes

## BMC in boosted variational inference

$$\text{RELBO}_{\mathcal{D}}(\mu_i, \Sigma_i) = \int \mathbb{E}[\log \bar{g}_{\mathcal{D}}(\theta)] \mathcal{N}(\theta | \mu_i, \Sigma_i) d\theta - \int \log(q_{i-1}(\theta)) \mathcal{N}(\theta | \mu_i, \Sigma_i) d\theta + \frac{\lambda}{4} \log |\Sigma_i|$$

$$\mathcal{L}_{i,\mathcal{D}}(\mathbf{w}) = \int \log \bar{g}_{\mathcal{D}}(\theta) ((1 - w_i) q_{i-1}(\theta) + w_i f_i(\theta)) d\theta - \int \log((1 - w_i) q_{i-1}(\theta) + w_i f_i(\theta)) ((1 - w_i) q_{i-1}(\theta) + w_i f_i(\theta)) d\theta$$

## Practical considerations

- RELBO stabilization

$$\text{RELBO}_{\mathcal{D}}^{\delta_D}(\mu_i, \Sigma_i) = \int \log \left( \frac{r_{\mathcal{D}}(\theta)}{q_{i-1}(\theta) + \delta_D} \right) \mathcal{N}(\theta; \mu_i, \Sigma_i) d\theta + \log |\Sigma_i|.$$

- Output scaling

$$\tilde{y}_i = (y_i - m_y)/\sigma_y, \tilde{\mathcal{D}} = \{x_i, \tilde{y}_i\}, \sigma_y \log g_{\tilde{\mathcal{D}}}(x) + \mu_y$$

- Component pruning: discard negligible components
- Initialization: either large covariance or maximize ELBO for first Gaussian component.
- Mean function:  $m(\theta) = C$  found to be more stable.

## Practical considerations

- Periodic joint parameter updating: sometimes maximize  $\mathbb{E}[\mathcal{L}_{\mathcal{D}}(\lambda)]$  for all parameters in  $\sum_{i=1}^k w_k \mathcal{N}(\theta; \mu_k, \Sigma_k)$ .
- Product of Matern kernels:

$$k_{\text{PMat},\nu}(x, x'; \theta, l) = \theta \prod_{d=1}^D k_{\text{Matern},\nu}(|x_i - x'_i|; l_d).$$

Is integrated in BVBMCMC by Gauss-Hermite quadrature. Found to be more stable than the SQE kernel.

- More acquisition functions:

$$\alpha_{\text{MMLT}}^{\mathcal{D}}(x_{m+1}) = e^{2m_{\mathcal{D}}(x) + k_{\mathcal{D}}(x,x)} \left( e^{k_{\mathcal{D}}(x,x')} - 1 \right).$$

$$\alpha_{\text{MMLT}_p}^{\mathcal{D}}(x_{m+1}) = e^{2m_{\mathcal{D}}(x) + k_{\mathcal{D}}(x,x)} \left( e^{k_{\mathcal{D}}(x,x')} - 1 \right) q_k(\theta_{N+1}; \lambda)^2.$$

## Usage of BVBMC package

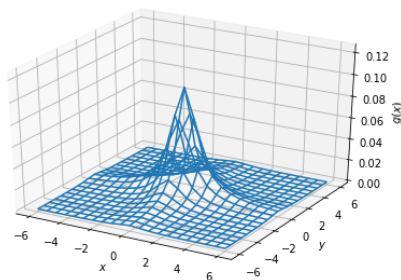
```

1  #Import necessary packages
2  import torch #PyTorch package
3  from variational_boosting_bmc import VariationalBoosting #BVBMC package
4
5  #Approximating unnormalized 2-d Cauchy
6  def logjoint(theta):
7      return torch.sum(-torch.log(1+theta**2))
8
9  #Set up parameters
10 dim=2 #Dimension of problem
11 samples = torch.randn(20,dim) #Initial samples
12 mu0 = torch.zeros(dim) #Initial mean
13 cov0 = 20.0*torch.ones(dim) #Initial covariance
14 acquisition = "prospective" #Acquisition function
15
16 #Initialize algorithm
17 vb = VariationalBoosting(dim, logjoint, samples, mu0, cov0)
18 vb.optimize_bmc_model() #Optimize GP model
19 vb.update_full() #Fit first component
20
21 #Training loop
22 for i in range(100):
23     _ = vb.update() #Choose new boosting component
24     vb.update_bmcmodel(acquisition=acquisition) #Choose new evaluation
25     vb.cutweights(1e-3) #Weights pruning
26     if ((i+1)%20) == 0:
27         vb.update_full(cutoff=1e-3) #Joint parameter updating
28
29 vb.save_distrib("finaldistrib") #Save distribution
30

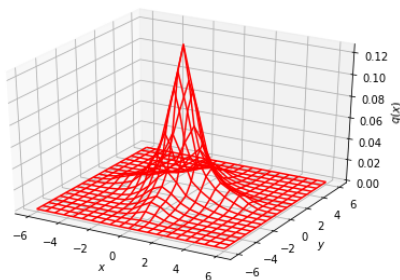
```



## Result from above code



True density



Estimated density

## BVBMC package

Open source Python package, that can be found in

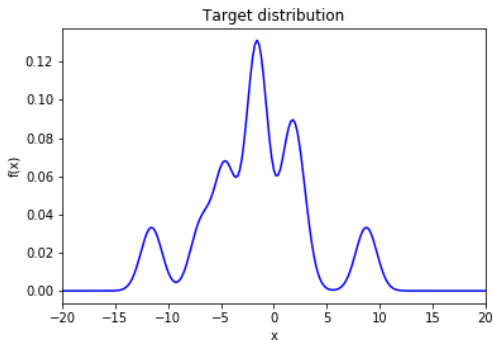
<https://github.com/DFNaiff/BVBMC>. Still lacks documentation (to be fixed soon).

Since it may (and probably will) undergo changes, code specific to this work can be found in <https://github.com/DFNaiff/Dissertation>.

## Implementation

Implementation of BVBMC package is heavily dependent on PyTorch. Due to the variety of inner optimizers, various gradient calculations are required. Automatic differentiation in PyTorch makes this process much more concise and less error prone.

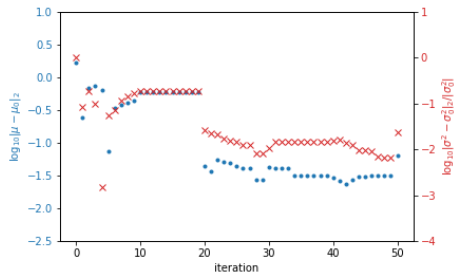
## 1-d mixture of Gaussians



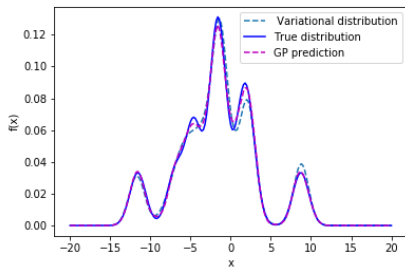
$$f(x) = \sum_{i=1}^{12} w_i \mathcal{N}(x; \mu_i, \sigma_i^2),$$

$$w_i = \frac{1}{12}, \mu_i \sim \mathcal{N}(0, \sqrt{5}), \sigma_i^2 = 1.$$

# Kernel performance

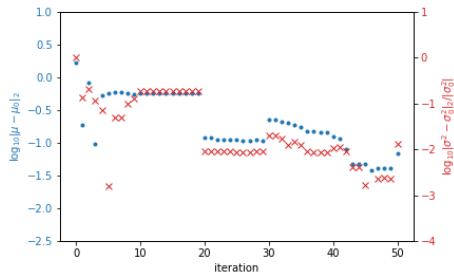


$k_{\text{PMat},5/2}$ , moments.

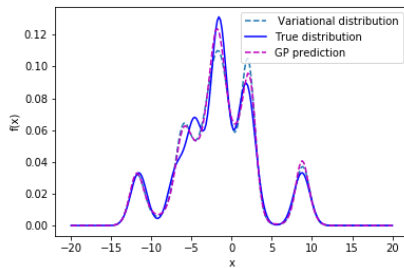


$k_{\text{PMat},5/2}$ , final result.

# Kernel performance

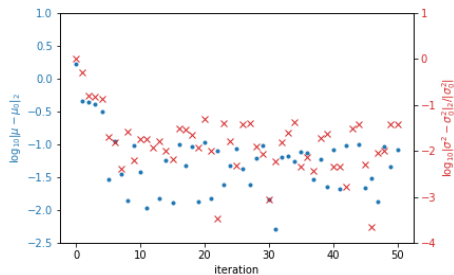


$k_{\text{SQE}}$ , moments.

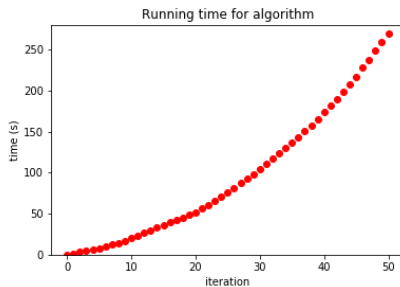


$k_{\text{SQE}}$ , final result.

# Training routine

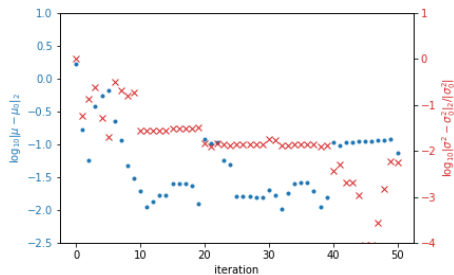


Routine A, moments

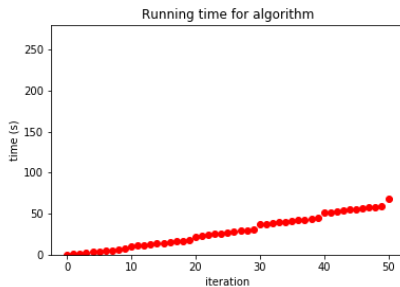


Routine A, running time.

# Training routine

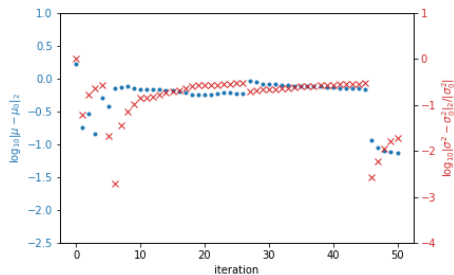


Routine B, moments.

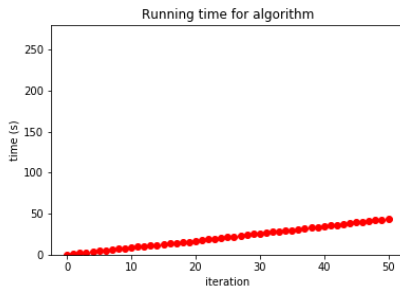


Routine B, running time.

# Training routine



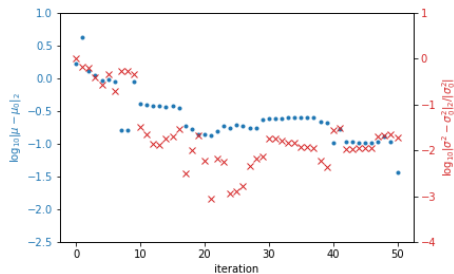
Routine C, moments.



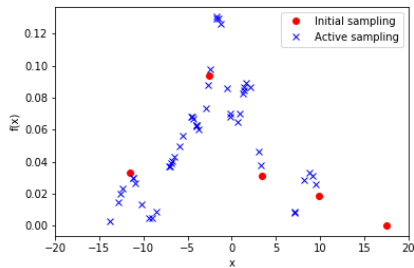
Routine C, running time.



## Active evaluation

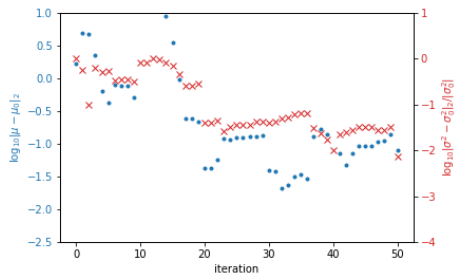


PROP, moments.

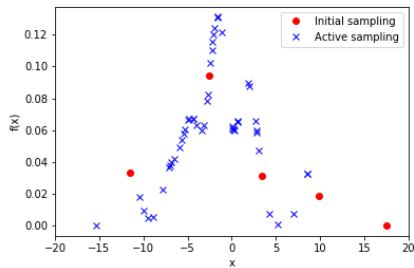


PROP, sampling.

## Active evaluation



MMLT, moments.



MMLT, sampling.

## N-d toy examples

- *Lumpy*

$$f(x) = \sum_{i=1}^{12} w_i \mathcal{N}(x; \mu_i, \Sigma_i),$$

$$(w_1, \dots, w_{12}) \sim \text{Dir}(1, \dots, 1), \mu_i \sim \text{Unif}([0, 1]^D), \\ \Sigma = \text{diag}(\sigma_1^2, \dots, \sigma_n^2), \sigma_i^2 \sim \text{Unif}(0.2, 0.6).$$

- *Cigar*

$$f(x) = \mathcal{N}(x; 0, \Sigma),$$

$$\Sigma = Q \Lambda Q^T, \Lambda = (10.0, 0.1, \dots, 0.1), Q \sim \text{Unif}(SO(D)).$$

- *Student-t*

$$f(x) = \prod_{d=1}^D \mathcal{T}(x_d; \nu_d),$$

$$\nu_d \sim \text{Unif}(2.5, 2 + 0.5D).$$

## N-d toy examples

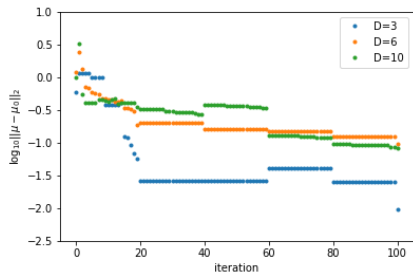
For each case, dimensions  $D = 2, 6, 10$  were tested, and the BVBM algorithm was run for 100 iterations, with  $10D$  initial samples. The GP kernel used were  $k_{\text{PMat}, \nu=2.5}$ , with active evaluation at each iteration, according to an acquisition function randomly chosen between the pair  $(\alpha_{\text{PROP}}, \alpha_{\text{MLT}})$ . Every 20 steps, joint parameter updating was done, and pruning was done at each iteration, with  $\beta = 10^{-3}$ .

## Comparison with VBMC

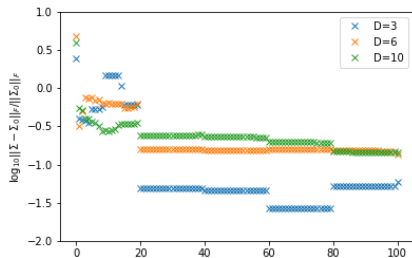
	Lumpy		Cigar	
	BVBMC	VBMC	BVBMC	VBMC
D=2	$3.12 \times 10^{-3}$	$6.5 \times 10^{-4}$	$8.12 \times 10^{-3}$	$2.1 \times 10^{-1}$
D=6	$6.59 \times 10^{-2}$	$3.5 \times 10^{-2}$	$5.56 \times 10^{-1}$	$1.07 \times 10^{-1}$
D=10	$1.19 \times 10^{-1}$	$4.2 \times 10^{-1}$	1.29	$1.0 \times 10^{-1}$

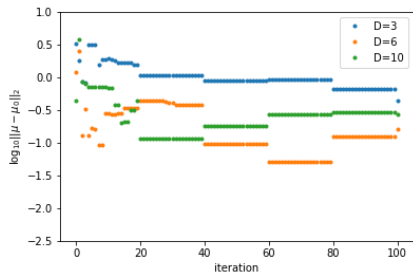
	Student-t	
	BVBMC	VBMC
D=2	$2.9 \times 10^{-1}$	$2.0 \times 10^{-3}$
D=6	$1.14 \times 10^{-1}$	$2.3 \times 10^{-1}$
D=10	$2.56 \times 10^{-1}$	$2.7 \times 10^{-1}$



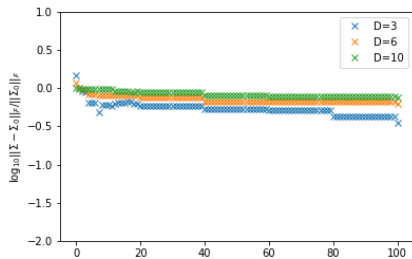
Lumpy, means convergence.



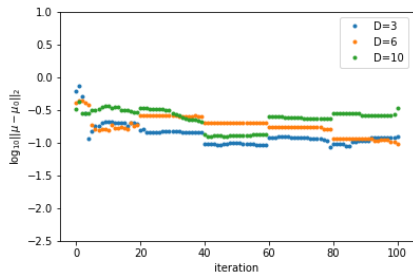
Lumpy, covariances convergence.



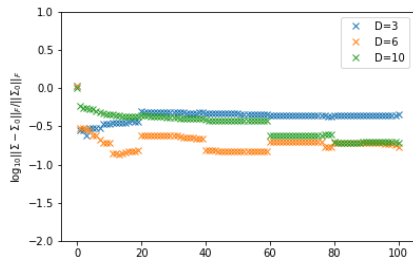
Cigar, means convergence.



Cigar, covariances convergence.



Student-t, means convergence.



Student-t, covariances convergence.



## Source problem

$$q(x, t) = q_0 \exp\left(-\frac{(x - x_0)^2}{2\rho^2}\right) \mathbf{1}_{[0, t_s)}(t).$$

$$\frac{\partial}{\partial t} u(x, t) = \frac{\partial^2}{\partial x^2} u(x, t) + q(x, t), \quad x \in (0, 1).$$

$$u(x, 0) = 0, \quad \frac{\partial}{\partial x} u(0, t) = \frac{\partial}{\partial x} u(1, t) = 0.$$

Objective: from measurements, estimate  $(x_0, q_0, t_s, \rho)$ .

## Likelihood model

For  $x_m = \{0, 1\}$ , measurements in  $t_m \in \{0.075, 0.15, 0.225, 0.3, 0.4\}$ .

$\mathcal{D} = \{\hat{u}(x_m, t_m)\}_{x_m \in \{0, 1\}, t_m \in T_M}$ .

$\hat{u}(x_m, t_m) = u(x_m, t_m) + \epsilon$ ,  $\epsilon \sim \mathcal{N}(0, \sigma^2)$ ,  $\sigma^2 \sim \text{InvGamma}(\alpha, \beta)$ .

$$p(\mathcal{D} | x_0, t_s, q_0, \rho) = \prod_{x_m \in \{0, 1\}, t_m \in T_M} \mathcal{T}(\hat{u}(x_m, t_m); u(x_m, t_m), \beta/\alpha, 2\alpha).$$

## Priors

$$p(x_0) = \text{Unif}(x_0; 0, 1)$$

$$p(t_s) = \text{Unif}(t_s; 0, 0.4)$$

$$p(q_0) = \text{HalfCauchy}(q_0; 10)$$

$$p(\rho) = \text{HalfCauchy}(\rho; 0.1)$$

## Warped problem in $\mathbb{R}^4$

$$x_0 = \text{sigmoid}(\tilde{x}_0)$$

$$t_s = 0.4 \times \text{sigmoid}(\tilde{t}_s)$$

$$q_0 = \exp(\tilde{q}_0)$$

$$\rho = \exp(\tilde{\rho}),$$

$$p(\tilde{x}_0, \tilde{t}_s, \tilde{q}_0, \tilde{\rho} | \mathcal{D}) \propto p(x_0, q_0, t_s, \rho | \mathcal{D}) \times \\ \text{sigmoid}'(\tilde{x}_0) \text{sigmoid}'(\tilde{t}_s) \exp(\tilde{q}_0) \exp(\tilde{\rho})$$

## Problem generation

A synthetic problem is considered with the true values being

$$x_0, t_s, q_0, \rho = 0.230, 0.300, 6.366, 0.050$$

The data was generated by solving the PDE by finite differences, and perturbing the measurements with by noise  $\mathcal{N}(0, 10^{-2})$ .

## Parameter estimation

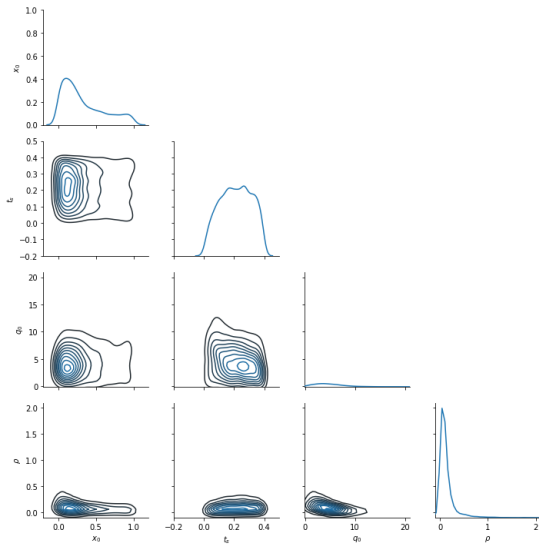
The likelihood is computed for each  $x_0, t_s, q_0, \rho$  by computing  $\hat{u}$  also by finite differences.

The BVBMC algorithm is applied to the problem, with a total of 180 evaluations. It was compared to the EMCEE algorithm, used in astrophysics, and parameters are estimated by their posterior calculated means.

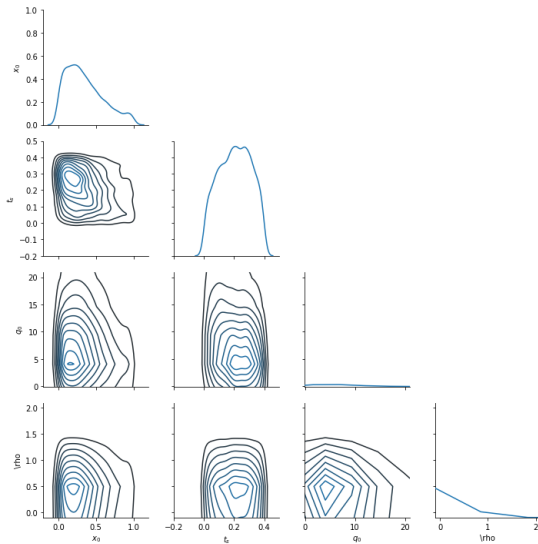
## True values and estimations

	$x_0$	$t_s$	$q_0$	$\rho$
True	0.230	0.300	6.366	0.050
BVBMC	0.328	0.213	5.435	0.140
EMCEE	0.352	0.206	10.228	0.218

## KDE for BVBMCMC solution



## KDE for EMCEE solution



## Challenges

- Boosted Variational Bayesian Monte Carlo is a "new" approach. As such, it remains to be seen in which cases it is best to use it.
- Posteriors in  $\mathbb{R}^D$  are limited, and the warping approach is clumsy. How can BVBMC be extended to a larger class of domains? Probably the reparameterization trick will have to be used.
- How can this approach be extended to pseudo-marginals?
- Is there a way to incorporate Sparse Gaussian Process here? The author has tried to do this, although he wasn't successful.

## Conclusion

The method presented in this work, although still immature, has shown promise for use in Bayesian inference, where the likelihood function is expensive to evaluate, that are common in inverse problems.

The associated package in <https://github.com/DFNaiff/BVBMC>, built on top of PyTorch, is intended to be easy to use, so a practitioner can quickly employ it in their own problems, if they wish so.