A Comparaison of Approximation Methods For Gaussian Process Regression

Arnaud Robert *
RIKEN Supervisor: Emtiyaz Khan
EPFL Supervisor: Matthias Grossglauser

May 10, 2017

*Address : 1, chemin de passe-loup, 1255 Veyrier, Switzerland

Contents

1 Summary **4**

2 Background **5**
2.1 Gaussian Process Regression (GPR) . . . 5
2.1.1 Posterior inference . . . 5
2.1.2 Hyperparameters learning . . . 6
2.1.3 Computational Challenges . . . 6
2.2 Covariance Functions . . . 6
2.2.1 Square Exponential (SE) Covariance Function . . . 6
2.2.2 Square Exponential with Automatic Relevance Determination (SEard) Covariance Function . . . 7
2.2.3 Periodic Covariance Function . . . 7
2.2.4 Linear Covariance Function . . . 7
2.3 Performance Measures . . . 7
2.3.1 Root Mean Square Error (rmse) . . . 7
2.3.2 Mean Predictive Probabilities . . . 8
2.3.3 Negative Log Marginal Likelihood . . . 8
2.4 Variational Inference . . . 8
2.4.1 Mean-Field Variational Inference . . . 9
2.4.2 Stochastic variational Inference (SVI . . . 9

3 Experimental set up **11**
3.1 Datasets . . . 11
3.2 Preprocessing . . . 11

4 Posterior Inference **12**
4.1 Low-rank Approximation of the Kernel . . . 12
4.1.1 Description . . . 12
4.1.2 Experimental results . . . 12
4.1.3 Details . . . 13
4.2 Subsets of Data Methods . . . 16
4.2.1 Description . . . 16
4.2.2 Experimental Results . . . 17
4.2.3 Details . . . 18

5 Hyperparameters Learning **19**
5.1 Description . . . 19
5.2 Experimental Results . . . 21
5.3 Details . . . 23

6 Conclusion **25**

7 Annexes **27**
7.1 Gaussian Identities 27
7.1.1 Trace trick 27
7.1.2 Kullback-Leiber Divergence for Multivariate Gaussian 27
7.2 Matrix operation 27
7.2.1 Matrix derivatives 27
7.2.2 Matrix Inversion Lemma 27
7.2.3 Matrix Determinant Lemma 27

8 Appendix **28**

1 Summary

Gaussian Process Regression (GPR) ([1], Chapter 2) is a probabilistic model for regression. Its ability to quantify uncertainty in its predictions makes it a popular choice for regression. However GPR is computationally inefficient —in order to do predictions it requires $\mathcal{O}(N^3)$ computation time and $\mathcal{O}(N^2)$ memory, where N is the number of observations.

This thesis discusses approximation methods for GPR. We compare methods in two different tasks: posterior inference and hyperparameters learning. We show comparisons on 9 different datasets whose size range from 300 to 35000 observations.

In the first part of the thesis we discuss approximation methods for inference. We first compare the following 3 different approximation methods that use a low-rank approximation of the GP kernel: Nyström approximation [6], Subset of Regressors (SoR) method [10] and Deterministic Training Conditional (DTC) method [11]. Those methods reduce the computational time requirement to $\mathcal{O}(m^2N)$ and the memory requirements to $\mathcal{O}(mN)$, where m is a constant such that $m < N$. Our results show that DTC systematically produce better results than the other two methods. We then compare two methods that use subsets of data: Subset of Data (SoD) ([1], Chapter 8) and Conjugate Variational Inference (CVI) [9]. We show that CVI performs, in general, better than SoD.

In the second part, we compare 3 different methods for learning: SoD, variational inference for DTC [3] (referred to as varDTC) and stochastic variational inference on DTC [4] (referred to as SVI-GP). The computational time required by varDTC is $\mathcal{O}(m^2N)$ while the computational time required by SoD and SVI-GP is $\mathcal{O}(m^3)$. We show in our experiments that SVI-GP and SoD have a much lower cost per iteration than varDTC, but SVI-GP is sensitive to initialization and noise in the stochastic gradients. SVI-GP and SoD have comparable performance on medium size datasets but the former is expected to work better on large datasets.

2 Background

2.1 Gaussian Process Regression (GPR)

Given an input $\mathbf{x}_i$ and its associated target value y_i, GPR models the deterministic function f that maps $\mathbf{x}_i$ to y_i, assuming that the target value is corrupted by random noise:

$$y_i = f(\mathbf{x}_i) + \epsilon \tag{1}$$

where $\epsilon \sim \mathcal{N}(0, \sigma^2)$.

Due to the property of Gaussian Process prior, any finite set of function evaluations $\mathbf{f} = f(\mathbf{X})$ follows a multivariate Gaussian distribution, where $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^N$ is a set of datapoints. We refer to $\mathbf{f}$ as latent functions (and sometimes as hidden variables or latent variables). We specify a prior over these functions : $p(\mathbf{f}) = \mathcal{N}(\mathbf{f}|0, \mathbf{K}_{ff})$. The prior is a Gaussian distribution with mean 0 and covariance $\mathbf{K}_{ff}$, where $\mathbf{K}_{ff}$ is a kernel function governed by some hyperarameters $\boldsymbol{\theta}$ (see section 2.2 for examples or [1] for detailed explanations on hyperparameters).

Given an unseen (test) datapoint $\mathbf{x}_*$ we express the following joint model:

$$\begin{bmatrix} \mathbf{y} \\ f_* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} \mathbf{K}_{ff} + \sigma_n^2 \mathbf{I} & \mathbf{K}_{f*} \\ \mathbf{K}_{*f} & k_{**} \end{bmatrix}\right) \tag{2}$$

where f_* represent the function evaluation of $f(x_*)$, $\mathbf{K}_{ff}$ is the covariance matrix of the training set, $\mathbf{K}_{f*}$ is the covariance between the training and the test set and k_{**} is the variance of the test point $\mathbf{x}_*$.

GPR regression can be decomposed in two phases : inference of the posterior distribution and learning of the hyperparameters θ.

2.1.1 Posterior inference

Making predictions corresponds to finding the function evaluation of our test datapoints: $f(\mathbf{x}_*)$. This is expressed by the following posterior distribution:

$$p(\mathbf{f}_*|\mathbf{y}) = \frac{p(\mathbf{f}_*, \mathbf{y})}{\int p(\mathbf{f}_*, \mathbf{y}) d\mathbf{f}_*}. \tag{3}$$

Since all distributions involved are Gaussian, the posterior distribution is also a Gaussian distribution :

$$p(\mathbf{f}_*|\mathbf{y}) \sim \mathcal{N}(\bar{\mathbf{f}}_*, cov(\mathbf{f}_*)), \tag{4}$$

where

$$\bar{\mathbf{f}}_* = \mathbf{k}_{*f}(\mathbf{K}_{ff} + \sigma^2 \mathbf{I})^{-1} \mathbf{y}$$
$$cov(\mathbf{f}_*) = k_{**} - \mathbf{k}_{*f}(\mathbf{K}_{ff} + \sigma^2 \mathbf{I})^{-1} \mathbf{k}_{f*}$$

We can obtain these equations by applying the identity given in eq. (60).

2.1.2 Hyperparameters learning

GPR is parameterized by several hyperparameters : the noise variance σ^2 and hyperparameters that define the kernel function denoted by $\boldsymbol{\theta}$ (more details on kernel parameters in section 2.2). In order to find good setting of these hyperparameters we maximize the marginal likelihood:

$$p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) = -\frac{1}{2}\mathbf{y}^T(\mathbf{K} + \sigma^2\mathbf{I})^{-1} - \frac{1}{2}\log|\mathbf{K} + \sigma^2\mathbf{I}| - \frac{n}{2}\log(2\pi) \tag{5}$$

We maximize the marginal likelihood by taking steps in the direction of its gradient:

$$\frac{\partial}{\partial\theta_j}\log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) = \frac{1}{2}\mathrm{Tr}\left[(\boldsymbol{\alpha}\boldsymbol{\alpha}^T - \mathbf{K}^{-1})\frac{\partial\mathbf{K}}{\partial\theta_j}\right] \tag{6}$$

2.1.3 Computational Challenges

In inference of the posterior we are required to compute the inverse $(\mathbf{K} + \sigma^2\mathbf{I})^{-1}$. Since the size of this matrix is $N \times N$ the computation time required is $\mathcal{O}(N^3)$. In learning we are required to compute the inverse $\mathbf{K}^{-1}$, and the product of two $N \times N$ matrices. These operations require $\mathcal{O}(N^3)$ computation time.

2.2 Covariance Functions

The covariance function plays a critical role in GP, we use them to encode our prior beliefs about the data such as smoothness or periodicity.

2.2.1 Square Exponential (SE) Covariance Function

This covariance function is governed by two hyperparameters $\boldsymbol{\theta} = \{l, \sigma^2\}$ that we will call respectively characteristic length scale and signal variance.

The formulation of the SE covariance matrix is as follow :

$$k_{SE}(r) = \sigma_f \exp\left(\frac{r^2}{2l^2}\right) \tag{7}$$

where $r = |\mathbf{x} - \mathbf{x}'|$.

This covariance function describe the following behavior: the covariance between two points (for a fixed length scale l) decrease according to a Gaussian function with zero mean and a standard deviation of l. If l is small only points that are close to each other will be correlated. On the other hand if l is large then even points that are far from each other will be correlated. The magnitude of the correlation is governed by the signal variance parameter σ_f.

There are several tools to describe the behavior of a kernel, see GP [1] Chapter 4, Mean Square (MS) differentiability define the smoothness of the kernel. It can be seen that SE kernel is infinitely MS differentiable, hence imposing a strong smoothness assumption.

2.2.2 Square Exponential with Automatic Relevance Determination (SEard) Covariance Function

One of the unrealistic assumption made in SE kernel is that covariance decays at the same pace in any dimension of the input space. This is not realistic, in practice some inputs are more relevant than the others and should have larger characteristic length scale. This is corrected by using SEard covariance function which allows one characteristic length scale per dimension of the input space:

$$k(\mathbf{x}, \mathbf{x}')_{SEard} = \sigma_f \exp\left(- \frac{(\mathbf{x} - \mathbf{x}')^T \mathbf{\Sigma}_l^{-1} (\mathbf{x} - \mathbf{x}')}{2} \right), \tag{8}$$

where Σ_l is a diagonal matrix that contains the D characteristic length scale on its diagonal. SEard is governed by $D + 1$ hyperparameters $\boldsymbol{\theta} = \{\Sigma_{ll}, \sigma^2\}$.

2.2.3 Periodic Covariance Function

Periodic kernels capture periodic behavior of the data, we express them as follow :

$$k_{periodic}(\mathbf{x}, \mathbf{x}') = k_{SE}(\mathbf{u}(\mathbf{x}), \mathbf{u}(\mathbf{x}')), \tag{9}$$

where $\mathbf{u}(\mathbf{x}) = [sin(\frac{\pi \mathbf{x}}{p}), cos(\frac{\pi \mathbf{x}}{p})]$. The hyperparameter p governs the periodicity of the covariance function. Here we wrap the periodic component into an SEiso kernel function so we also have to learn the hyper-parameters of the SE covariance function (σ_f and l) and we end up with the following set of hyperparameters $\boldsymbol{\theta} = \{l, \sigma^2, p\}$

2.2.4 Linear Covariance Function

This covariance function is not stationary, it is defined as follow:

$$k_{lin}(\mathbf{x}, \mathbf{x}') = \mathbf{x} \mathbf{\Sigma}_l^{-1} \mathbf{x}', \tag{10}$$

where $\mathbf{\Sigma}_l$ is a diagonal matrix with the D characteristic length scale parameters on its diagonal, the D hyperparameters of this kernel are $\boldsymbol{\theta} = \{\Sigma_{ll}\}$.

2.3 Performance Measures

To compare various algorithms, we will use the following performance measures (also called loss functions).

2.3.1 Root Mean Square Error (rmse)

RMSE is defined as follows:

$$rmse = \sqrt{\frac{1}{N_{te}} \sum_{i=1}^{N_{te}} (\mathbf{y}_i^{te} - \hat{\mathbf{y}}_i)^2} \tag{11}$$

where N_{te} is the number of datapoints in the test set, $\mathbf{y}_i^{te}$ are the true output values with $i \in \{1, ..., N_{te}\}$ and $\hat{\mathbf{y}}_i$ is the corresponding predicted value.

2.3.2 Mean Predictive Probabilities

The second loss function used is simply the mean of the log probability of the likelihood of $\mathbf{y}^{te}$:

$$lp = \frac{1}{N_{te}} \sum_{i=1}^{N_{te}} - \log p(y_i^{te}) \tag{12}$$

where $p(y_i^{te}) = \mathcal{N}(y_i^{te}|\mu_i, \sigma_i^2)$. And the two parameters of the Gaussian distribution are defined as follow for Gaussian Process Regression:

$$\begin{aligned} \mu_i &= \mathbf{k}(\mathbf{x}_i^{te}, \mathbf{X})[\mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma^2\mathbf{I}]^{-1}\mathbf{y}, \\ \sigma_i^2 &= k(\mathbf{x}_i^{te}, \mathbf{x}_i^{te}) - \mathbf{k}(\mathbf{x}_i^{te}, \mathbf{X})[\mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma^2\mathbf{I}]^{-1}\mathbf{k}(\mathbf{X}, \mathbf{x}_i^{te}), \end{aligned} \tag{13}$$

where x_i^{te} is the ith input of the test data and $\mathbf{X}$ denotes the entire training data. $\mathbf{K}(\mathbf{X}, \mathbf{X})$ and $\mathbf{k}(\mathbf{x}_i, \mathbf{X})$ are the kernel functions (see section 2.2).

2.3.3 Negative Log Marginal Likelihood

The last loss function used is the Negative Log Marginal Likelihood, where the Marginal Likelihood on training data $\mathbf{y}$ is given by :

$$\log\, p(\mathbf{y}|\mathbf{X}, \Theta) = -\frac{1}{2}\mathbf{y}^T(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{y} - \frac{1}{2}\log\,|(\mathbf{K} + \sigma^2\mathbf{I})| - \frac{n}{2}log\,2\pi \tag{14}$$

Despite the fact that this error is based on the training data, the second term in equation (14) penalize for model complexity, hence prevent from overfitting (see [1], page 113).

We should keep in mind the main differences between our loss functions:

- While the Negative Log Marginal Likelihood is only based on the training data, the other two loss functions are based on the test data.
- While RMSE take only into account the mean of the predictions, the log predictive probabilities take also into account the variance of the predictions.

2.4 Variational Inference

As we have already seen in section 2.1, when dealing with probabilistic model, the distribution of main interest is the posterior distribution.

$$p(\mathbf{z}|\mathbf{y}) = \frac{p(\mathbf{y}, \mathbf{z})}{\int p(\mathbf{y}, \mathbf{z})d\mathbf{z}} \tag{15}$$

where $\mathbf{z}$ is any latent function. Exact posterior might be to difficult to compute, due to the integral in the denominator. Instead of computing the exact posterior we will approximate it. There is two kind of approximations, random approximation such as MCMC or deterministic approximation. In this section we focus on a deterministic approximation method called variational inference. We present two different versions of it, the Mean-Field Variational Inferencea and the Stochastic Variational Inference. The key idea behind this two methods is that they transform inference problem into an optimization problem.

2.4.1 Mean-Field Variational Inference

In mean-field variational inference we introduce a distribution $q_\theta(\mathbf{z}|\mathbf{y})$ governed by some parameters θ, that aims to approximate $p(\mathbf{z}|\mathbf{y})$. The only assumption made in mean field variational inference is that the latent functions are independent from each other.
Then, we optimize the variational parameters θ to get a solution as close as possible of the true posterior distribution, closeness is defined in terms of Kullback-Leibler (KL) divergence.

Minimizing the the KL divergence is equivalent to maximizing the evidence lower bound (ELBO), a lower bound on the logarithm of the marginal probability of the observations $\log p(\mathbf{y})$.

We recall below the derivation of the lower bound: (see [5], equation (8))

$$\begin{aligned}
\log(p(\mathbf{y})) &= \log \int p(\mathbf{y}, \mathbf{z}) d\mathbf{z} && (16)\\
&= \log \int p(\mathbf{y}, \mathbf{z}) \frac{q(\mathbf{z})}{q(\mathbf{z})} d\mathbf{z} && (17)\\
&= E_q \left[\frac{p(\mathbf{y}, \mathbf{z})}{q(\mathbf{z})} \right] && (18)\\
&\geq E_q[\log p(\mathbf{y}, \mathbf{z})] - E_q[\log q(\mathbf{z})] && (19)
\end{aligned}$$

where we simply applied Jensen's inequality to go from equation (18) to (19).

2.4.2 Stochastic variational Inference (SVI

SVI turns inference into a stochastic optimization problem.
Stochastic gradient descent (SGD) follow direction of gradient computed on small batch of datapoints. Computing the gradient as follow allows us to have an approximation of the direction of the steepest ascent at a cheap computational price.

Saying that gradients point to the steepest ascent direction is equivalent to solve the following maximization problem.

$$\underset{d\lambda}{\operatorname{argmax}} f(\boldsymbol{\lambda} + d\boldsymbol{\lambda}) \quad \text{subject to } ||d\boldsymbol{\lambda}||^2 < \epsilon^2 \tag{20}$$

However, as illustrated in the small example below (taken from [5]), when optimizing probability distribution Euclidean distance is not a good metric.

> The distributions $\mathcal{N}(0, 10000)$ and $\mathcal{N}(10, 10000)$ are almost indistinguishable, and the Euclidean distance between their parameter vectors is 10. In contrast, the distributions $\mathcal{N}(0, 0.01)$ and $\mathcal{N}(0.1, 0.01)$ barely overlap, but this is not reflected in the Euclidean distance between their parameter vectors, which is only 0.1.

The problem is that Euclidean distance doesn't capture the geometry of the space described by the probability distribution.
SVI corrects this issue by taking derivatives with respect to its natural parameters. The Euclidean distance metric is replaced by the symmetric KL divergence D_{KL}^{sym}.

$$\underset{d\lambda}{\operatorname{argmax}} f(\boldsymbol{\lambda} + d\boldsymbol{\lambda}) \quad \text{subject to } D_{KL}^{sym}(\boldsymbol{\lambda}, \boldsymbol{\lambda} + d\boldsymbol{\lambda}) < \epsilon \tag{21}$$

where

$$D_{KL}^{sym}(\boldsymbol{\lambda}, \boldsymbol{\lambda}') = E_{\boldsymbol{\lambda}}\left[\log \frac{q(\mathbf{z}|\boldsymbol{\lambda})}{q(\mathbf{z}|\boldsymbol{\lambda}')}\right] + E_{\boldsymbol{\lambda}'}\left[\log \frac{q(\mathbf{z}|\boldsymbol{\lambda}')}{q(\mathbf{z}|\boldsymbol{\lambda}}\right] \tag{22}$$

Then the algorithm works as follow. It select a random subset of the datapoints, use them to compute a noisy estimate of the gradient of the ELBO, do a parameter update and repeat this operation until convergence.

Algorithm 1 SVI

loop:
  randomly select subset X_r.
  $\theta_{t+1} = (1 - r) * \theta_t + r * \frac{\partial \mathcal{L}(X_r, \theta_t)}{\partial \theta}$.
goto *loop*.

Where $\mathcal{L}$ is the ELBO and r is the step size.

Dataset Name	# Trains	# Tests	# Dimensions
Yacht Hydrodynamics	154	154	7
Housing	455	51	13
Concrete Compressive Strength	515	515	8
Airfoil Self-Noise	752	751	5
Abalone	3132	1045	8
kin8nm	4096	4096	8
pumadyn-8nm	4096	4096	8
Sarcos	44484	4900	21
CHEM	35984	35984	15

Table 1: Summary of the data sets characteristics.

3 Experimental set up

3.1 Datasets

For the experiment we use 9 different datasets in our experiments. Table 1 gives details about all the datasets.

- **Housing Dataset**: Each feature is a factor that explain housing price variation in a specific area and the output is the median-value of owner-occupied homes (in $1000).Refers to [12] for more details on the data set.
- **Airfoil Self-Noise Dataset**: Each feature describe characteristic of an aifoil and the corresponding output is the noise generated by this airfoil. Refer to [12] for more details on the data set.
- **Concrete Compressive Strength Dataset**: Each input variable from describes characteristic of concrete and the output variable is the compressive strength of the corresponding concrete. Refers to [13] for more details on the dataset. The dataset is available at [12]
- **Yacht Hydrodynamics Dataset**: The inputs describe the hull geometry of sailing yacht and the output is the resistance suffered. Refers to [12] for more details on the data set.
- **Abalone Dataset**: Use physical measurements such as height and shell weight to predict the age of abalone, a complete description of the attributes is available at [12].
- **Kin8nm, Sarcos, Pumadyn-8nm Dataset** are robotics datasets details on their respective attributes can be found in [12].

3.2 Preprocessing

For each dataset we use the following preprocessing scheme. We first apply a train-test split (see table 1 for proportion), then we compute the mean $\bar{\mathbf{x}}_{tr}$ and standard deviation $\mathbf{x}_{tr}^{sd}$ of each dimension

of the training set.
Then for each datapoint in training and testing set we apply the following operation :

$$\mathbf{x}_{new} = \frac{\mathbf{x} - \bar{\mathbf{x}}_{tr}}{\mathbf{x}_{tr}^{sd}} \tag{23}$$

Using only statistics from training set we avoid any leaking of information from the test data into our model. Normalizing the data confers nice property when taking gradient steps and also allows easier comparaison of performances across datasets.

4 Posterior Inference

4.1 Low-rank Approximation of the Kernel

4.1.1 Description

We compare the performance of three different approximation methods: Nyström approximation, SoD and DTC. Each method uses a low-rank approximation of the kernel matrix $\mathbf{K}_{ff} \approx \mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{K}_{uf}$, where $\mathbf{u}$ is a small set of m inducing variables, where $m << N$.

Each approximation method employs a different approximate joint model $p(\mathbf{f}, \mathbf{f}_*)$ as shown below (detailed derivation is given in 4.1.3).

$$\text{Nyström:} \quad q_{NYS}(\mathbf{f}, \mathbf{f}_*) = \mathcal{N}\left(0, \begin{bmatrix} \mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{K}_{uf} & \mathbf{K}_{f*} \\ \mathbf{K}_{*f} & \mathbf{K}_{**} \end{bmatrix}\right. \tag{24}$$

$$\text{DTC:} \quad q_{DTC}(\mathbf{f}, \mathbf{f}_*) = \mathcal{N}\left(0, \begin{bmatrix} \mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{K}_{uf} & \mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{K}_{u*} \\ \mathbf{K}_{*u}\mathbf{K}_{uu}^{-1}\mathbf{K}_{uf} & \mathbf{K}_{**} \end{bmatrix}\right. \tag{25}$$

$$\textit{SoR:} \quad q_{SoR}(\mathbf{f}, \mathbf{f}_*) = \mathcal{N}\left(0, \begin{bmatrix} \mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{K}_{uf} & \mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{K}_{u*} \\ \mathbf{K}_{*u}\mathbf{K}_{uu}^{-1}\mathbf{K}_{uf} & \mathbf{K}_{*u}\mathbf{K}_{uu}^{-1}\mathbf{K}_{u*} \end{bmatrix}\right) \tag{26}$$

Nyström approximation simply replaces $\mathbf{K}_{ff}$ by its low rank approximation, without modifying the off-diagonal elements of the joint covariance. Note that there is no guarantee that the resulting joint covariance is positive definite. SoR applies the low-rank approximation on the covariance matrix of test points, while DTC model keeps it unchanged. We will see that DTC approach leads to a more realistic model of the variance. (Note that the predictive mean of both models are the same, see 4.1.3).

4.1.2 Experimental results

We compare the performances of models as a function of the number of inducing points. In each iteration, we form the set of inducing points by randomly sampling a new point from the training set, without replacement. We measure the performance using three different loss functions (as defined in Section 2.3): rmse, negative mean log probability and negative marginal likelihood. We repeat this experiment for 50 different random seeds. Note that we keep the test set fixed and only randomly select the inducing points.
We do this experiment on three different kernels: Square Exponential isotropic (SEiso) kernel,

Square Exponential automatic relevance determination and Linear isotropic (LINiso) kernel and on 5 different datasets, results for Housing dataset with SEiso and SEard kernels are given in Figure 1 and 2, and results for other datasets and kernels are shown in Appendix in Figures 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18.

We consistently observe that Nyström approximation performs poorly when the number of inducing points is small. This is also observed in [8]. RMSE curves of SoR model and DTC model shows that they have the same predictive mean as detailed in 4.1.3. However negative mean log probability curves consistently show that DTC model propose a more reasonable estimation of the variance.

4.1.3 Details

This section gives more detailed explanation about each model.

Subset of Regressors model:

Given a test point $\mathbf{x}_*$, SoR computes the following prediction: $f_* = \mathbf{K}_{*u}\mathbf{w}_u$. Where we have a weight w_{u_i} for each inducing point $\mathbf{X}_{u_i}$. The prior over the weight is $p(\mathbf{w}_u) \sim \mathcal{N}(0, \mathbf{K}_{uu}^{-1})$. To express this model as a GPR model, we need to find the prior distribution of the inducing variables $\mathbf{u}$, where $\mathbf{u} = f(\mathbf{X}_u)$. From the model definition we have that $\mathbf{u} = \mathbf{K}_{uu}\mathbf{w}_u$. We recover the prior on $\mathbf{u}$ observing the following :

$$\mathbb{E}[\mathbf{u}] = \mathbb{E}[\mathbf{K}_{uu}\mathbf{w}_u] = 0 \tag{27}$$

$$\mathbb{E}[\mathbf{u}\mathbf{u}^T] = \mathbb{E}[\mathbf{K}_{uu}\mathbf{w}_u\mathbf{w}_u^T\mathbf{K}_{uu}] = \mathbf{K}_{uu} \tag{28}$$

We can now substitute the weights $\mathbf{w}_u$ in SoR predictive expression by $\mathbf{K}_{uu}^{-1}\mathbf{u}$:

$$\mathbf{f}_* = \mathbf{K}_{*u}\mathbf{K}_{uu}^{-1}\mathbf{u}, \tag{29}$$

where $p(\mathbf{u}) \sim \mathcal{N}(0, \mathbf{K}_{uu})$.

This determinist relation between $\mathbf{u}$ and f_* allows us to express the following two conditional distributions:

$$p_{SoR}(\mathbf{f}|\mathbf{u}) = \mathcal{N}(\mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{u}, 0) \tag{30}$$

$$p_{SoR}(\mathbf{f}_*|\mathbf{u}) = \mathcal{N}(\mathbf{K}_{*u}\mathbf{K}_{uu}\mathbf{u}, 0) \tag{31}$$

Using the above equations we can formulate SoR joint model:

$$q_{SoR}(\mathbf{f}, \mathbf{f}_*) = \mathcal{N}\left(0, \begin{bmatrix} \mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{K}_{uf} & \mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{K}_{u*} \\ \mathbf{K}_{*u}\mathbf{K}_{uu}^{-1}\mathbf{K}_{uf} & \mathbf{K}_{*u}\mathbf{K}_{uu}^{-1}\mathbf{K}_{u*} \end{bmatrix}\right) \tag{32}$$

SoR predictive mean and variance can be expressed as follow:

$$p_{SoR}(\mathbf{f}_*|\mathbf{y}) \sim \mathcal{N}(\sigma^{-2}\mathbf{K}_{*u}\mathbf{\Sigma}\mathbf{K}_{uf}\mathbf{y}, \mathbf{K}_{*u}\mathbf{\Sigma}\mathbf{K}_{u*}) \tag{33}$$

where $\mathbf{\Sigma} = (\sigma^{-2}\mathbf{K}_{uf}\mathbf{K}_{fu} + \mathbf{K}_{uu})^{-1}$.

We note that the predictive mean of SoR is equivalent to the predictive mean of DTC eq. (38).

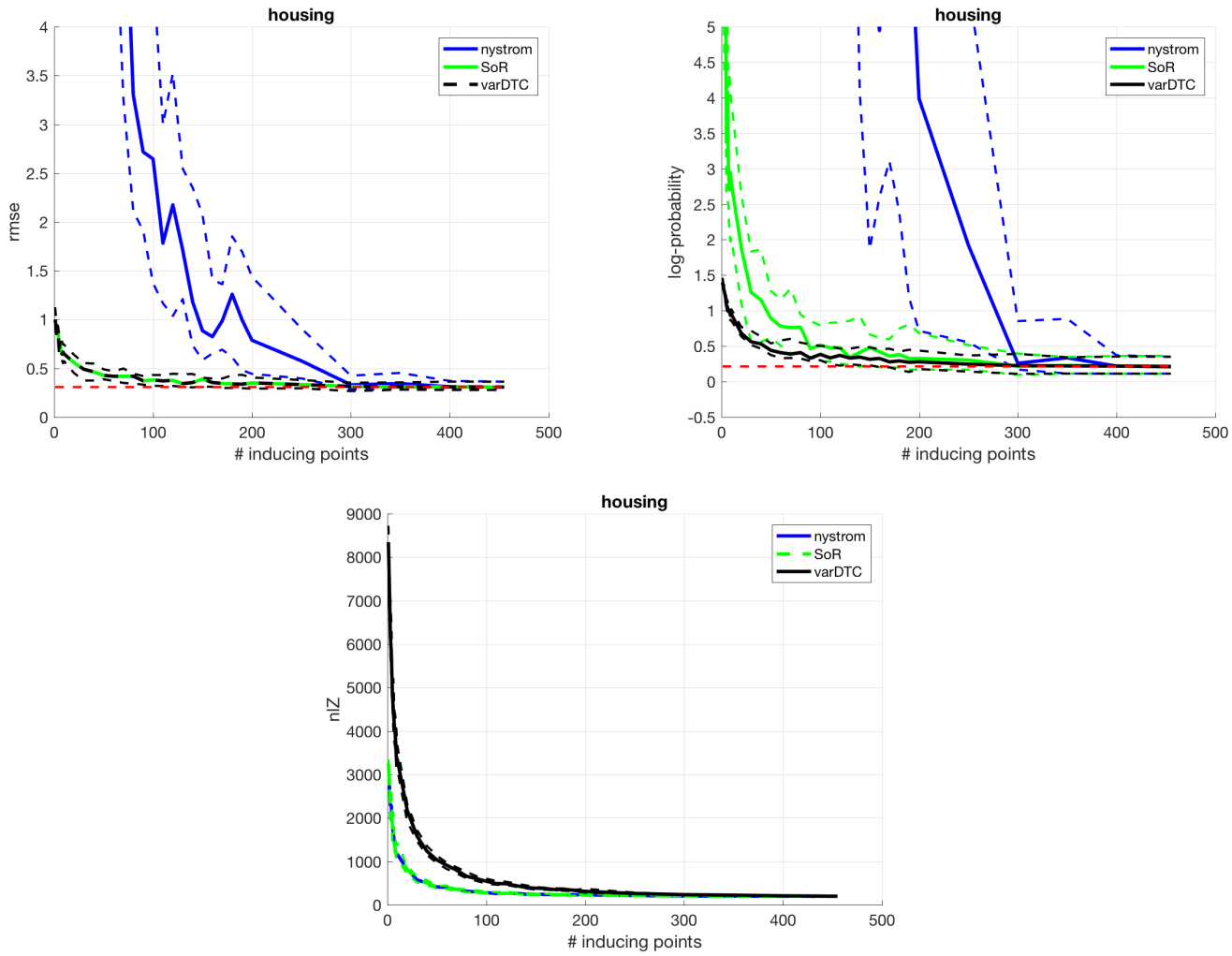


Figure 1: This figure shows the performance of the three low-rank approximation kernel algorithms as a function of the number of inducing points on **Housing** dataset. In this experiment the kernel used is the **SEiso** kernel. Each figure shows a different loss function. The red dotted line shows the performance of exact GP regression. Dashed line represent the 25th and 75th percentile while solid lines shows the median. We see that rmse curves of DTC and SoR are exactly the same, but DTC's mean log-probability curve is performing better than SoR and Nyström curves.

Deterministic Training Conditional Model:

DTC use the following projection $\mathbf{f} = \mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{u}$ to approximate the likelihood :

$$p_{DTC}(\mathbf{y}|\mathbf{u}) = \mathcal{N}(\mathbf{y}|\mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{u}, \sigma^2\mathbf{I}) \tag{34}$$

This approximation can be equivalently defined as using the exact likelihood but forcing the

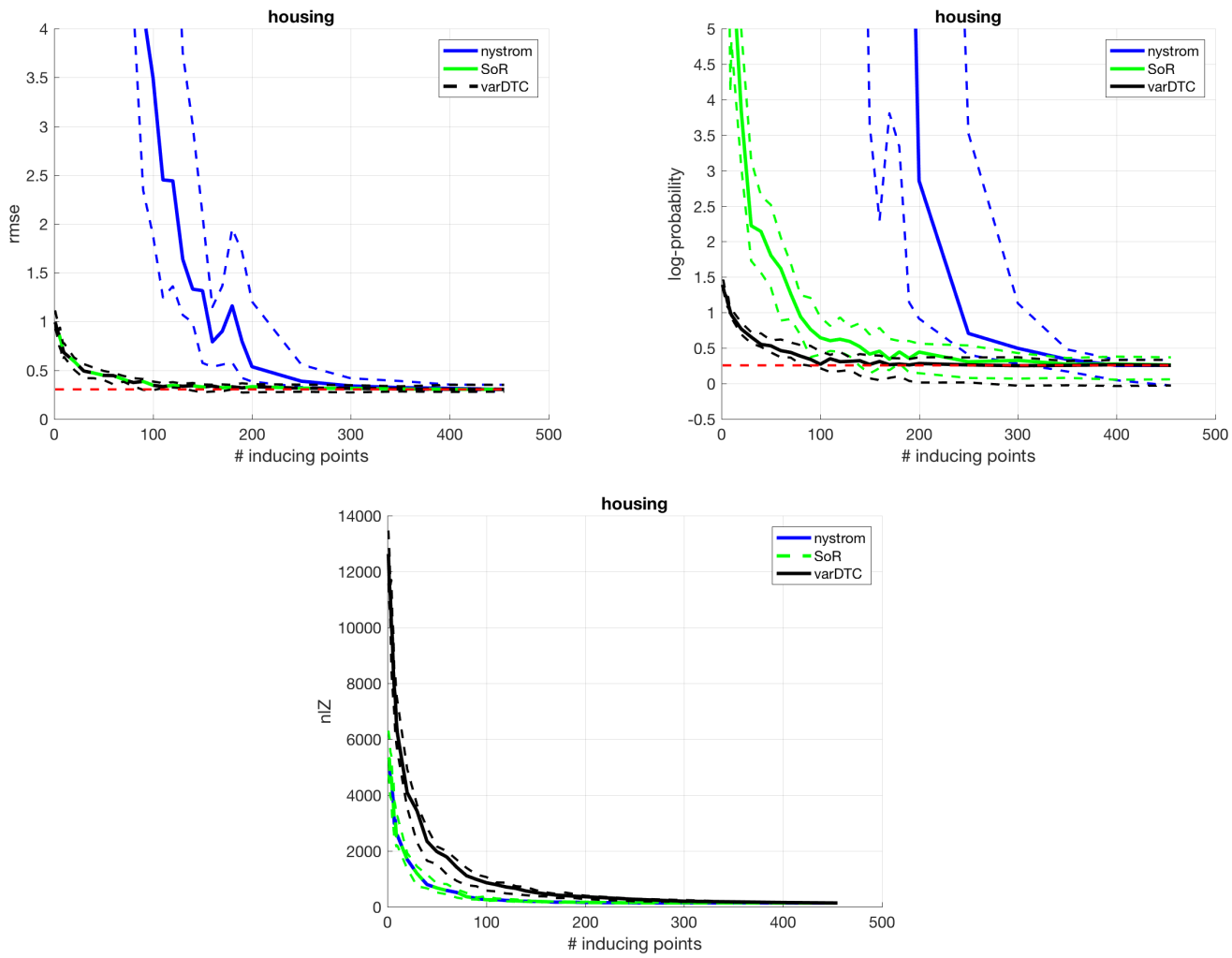


Figure 2: This figure shows the performance of the three low-rank approximation kernel algorithms as a function of the number of inducing points on **Housing** dataset. In this experiment the kernel used is the **SEard** kernel. Each figure shows a different loss function. The red dotted line shows the performance of exact GP regression. Dashed line represent the 25th and 75th percentile while solid lines shows the median. We see that rmse curves of DTC and SoR are exactly the same, but DTC's mean log-probability curve is performing better than SoR and Nyström curves.

following two conditional distributions :

$$p_{DTC}(\mathbf{f}|\mathbf{u}) = \mathcal{N}(\mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{u}, 0) \tag{35}$$

$$p_{DTC}(\mathbf{f}_*|\mathbf{u}) = \mathcal{N}(\mathbf{f}_*, 0) \tag{36}$$

It becomes easy now to express the following joint model:

$$q_{DTC}(\mathbf{f}, \mathbf{f}_*) = \mathcal{N}\left(0, \begin{bmatrix} \mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{K}_{uf} & \mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{K}_{u*} \\ \mathbf{K}_{*u}\mathbf{K}_{uu}^{-1}\mathbf{K}_{uf} & \mathbf{K}_{**} \end{bmatrix}\right) \tag{37}$$

Note that the only difference in DTC approximate prior and SoR approximate prior is the covariance of the test points.

DTC has the following predictive distribution:

$$p_{DTC}(\mathbf{f}_*|\mathbf{y}) \sim \mathcal{N}(\sigma^{-2}\mathbf{K}_{*u}\mathbf{\Sigma}\mathbf{K}_{uf}\mathbf{y}, \mathbf{K}_{**} - \mathbf{K}_{*u}\mathbf{K}_{uu}^{-1}\mathbf{K}_{u*} + \mathbf{K}_{*u}\mathbf{\Sigma}\mathbf{K}_{u*} \tag{38}$$

where $\mathbf{\Sigma} = (\sigma^{-2}\mathbf{K}_{uf}\mathbf{K}_{fu} + \mathbf{K}_{uu})^{-1}$.

Nyström approximation:
In Nyström GP we simply replace all occurrences of the training covariance matrix $\mathbf{K}_{ff}$ by a low rank approximation.

This gives us the following joint model :

$$q_{nys}(\mathbf{f}, \mathbf{f}_*) = \mathcal{N}\left(0, \begin{bmatrix} \mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{K}_{uf} & \mathbf{K}_{f*} \\ \mathbf{K}_{*f} & \mathbf{K}_{**} \end{bmatrix}\right) \tag{39}$$

Note that there is no guarantee that the covariance in eq. (39) is positive definite, enabling the model to produce negative predictive variances.

Finally, we rewrite the Nyström predictive distribution as follow :

$$p_{nys}(\mathbf{z}|\mathbf{y}) \sim \mathcal{N}(\bar{\mathbf{f}}_*, cov(\mathbf{f}_*)), \tag{40}$$

where,

$$\bar{\mathbf{f}}_* = \mathbf{k}_{*n}(\tilde{\mathbf{K}}_{ff} + \sigma^2\mathbf{I})^{-1}\mathbf{y}$$
$$cov(\mathbf{f}_*) = k_{**} - \mathbf{k}_{*n}(\tilde{\mathbf{K}}_{ff} + \sigma^2\mathbf{I})^{-1}\mathbf{k}n*$$

whith $\tilde{\mathbf{K}} = \mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{K}_{uf}$ is the low rank approximation.

4.2 Subsets of Data Methods

4.2.1 Description

In this experiment we consider CVI method and SoD method. They both maintain an active set I_t, which contains the subset of data seen at time t. SoD computes exact GPR posterior distribution on the subset I_t :

$$p(f_*|\mathbf{y}) = \prod_{i \in I_t} \mathcal{N}(y_i|f_i, \sigma^2)\mathcal{N}(\mathbf{f}_t|0, \mathbf{K}_{tt}) \tag{41}$$

where $\mathbf{K}_{tt}$ is the covariance matrix of datapoints in I_t and $\mathbf{f}_t$ correspond to function evaluation of the points contained in I_t.

CVI updates correspond to GPR inference, where each datapoint has its own noise variance. In each iteration CVI updates the variance of the selected datapoints and compute the following posterior distribution

$$p(f_*|\mathbf{y}) = \prod_{i \in I_t} \mathcal{N}(y_i|f_i, 1/\tilde{\lambda_{i,t}})\mathcal{N}(\mathbf{f}_t|0, \mathbf{K}_{tt}) \tag{42}$$

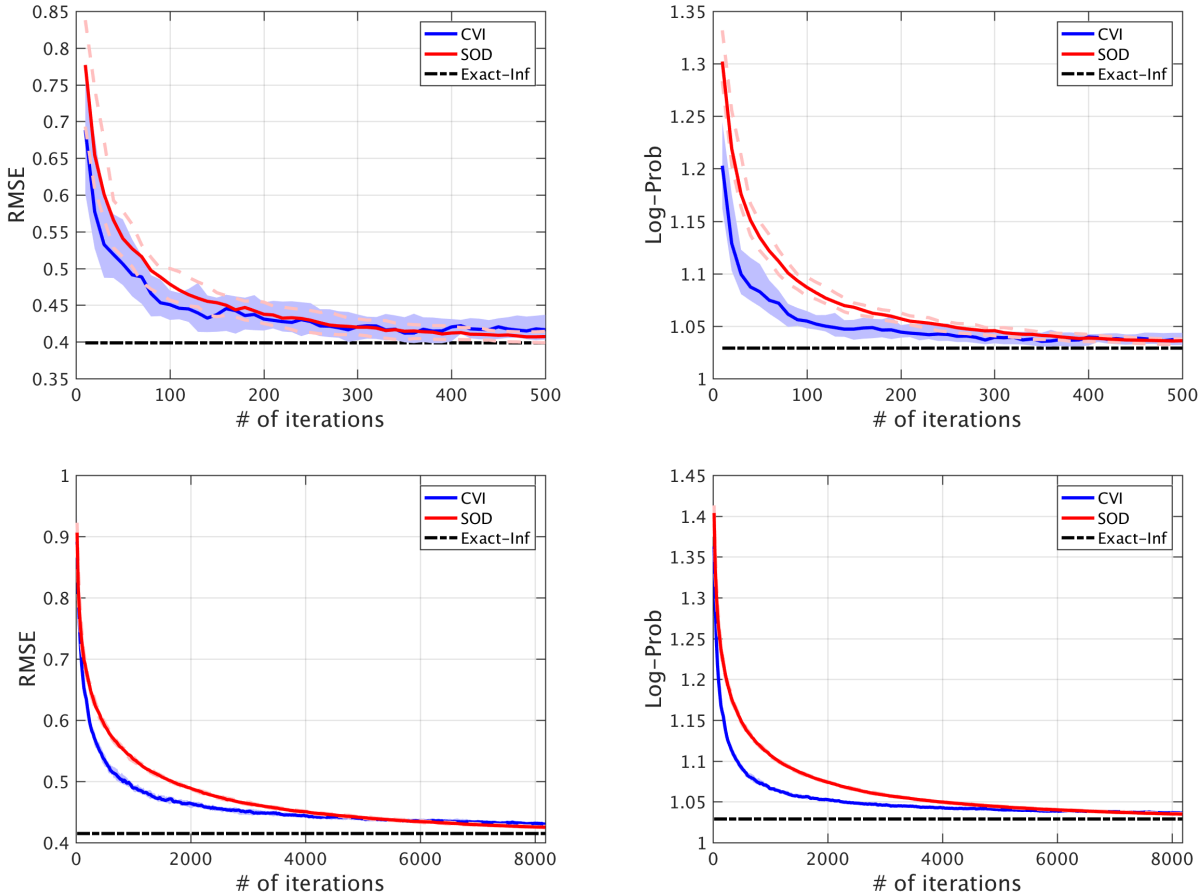
where $\tilde{\lambda}_{i,t} = (1 - r_t)\tilde{\lambda}_{i,t-1} + r_t \frac{N}{\sigma^2}$ for the index i selected at time t and $\tilde{\lambda}_{i,t} = (1 - r_t)\tilde{\lambda}_{i,t-1}$ for the rest, with $r_t = \beta_t/(1 + \beta_t)$ (more details on CVI updates are given in section 4.2.3).

In order to compare those two methods we shows the evolution of the 2 different loss functions: rmse and mean log-probability with respect to the number of iterations. One iteration in SoD consist in picking a datapoint at random with replacement. If the datapoint is not already included in the SoD active set, we insert it in $\mathbf{I}_t$. An iteration for CVI consists of an update for the parameter $\tilde{\boldsymbol{\lambda}}$. We run up to $t = 2N$ iterations. We expect to visit around 85% of the data examples which is enough for both algorithms to converge. After each iteration, we evaluate the loss functions and we repeat the experiments for 50 differents random seeds. Note that for both, CVI and SoD, we used the hyperparameters learned by exact GP, for CVI we fixed the step-size to $\beta = (1 + t)0.8$.

4.2.2 Experimental Results

Figure 3 shows the results of this experiment on Housing and Kin8nm dataset (results for other datasets are presented in Appendix in Figures 19 and 20). Left column compares the evolution of RMSE with versus the number of iterations, while the right column compares the evolution negative of log-probability.

We see that CVI consistently perform better than SoD reaching perfomance close to exact inference faster (Abalone dataset is an exception, see Figure 19 in Appendix). It is to be noted that, in none of these datasets, performance of SoD does not saturates as the amount of data is increased. This implies that the datasets are fairly small and more data will improve the performance of GP.



RMSE Negative Mean Log Probability

Figure 3: The first raw of plots shows the results obtained on **Housing** dataset (**253** training points), the second raw shows the results obtained on **Kin8nm** dataset (**4096** training points). The blue shaded area covers the 25th-75th percentiles of the CVI performance. The red dashed lines represent the 25th and 75th percentile of SoD performance. The black dashed line shows the performance of exact GPR.

4.2.3 Details

Subset of Data model:

SoD use a subset of data X_m of m datapoints and apply exact GPR on it. Its predictive distribution is given below.

$$p(\mathbf{f}_*|\mathbf{y}) \sim \mathcal{N}(\bar{\mathbf{f}_*}, cov(\mathbf{f}_*)), \tag{43}$$

where

$$\bar{\mathbf{f}_*} = \mathbf{k}_{*n}(\mathbf{K}_{mm} + \sigma^2\mathbf{I})^{-1}\mathbf{y}$$
$$cov(\mathbf{f}_*) = k_{**} - \mathbf{k}_{*m}(\mathbf{K}_{mm} + \sigma^2\mathbf{I})^{-1}\mathbf{k}m*,$$

where $\mathbf{K}_{mm}$ is the $m \times m$ covariance matrix between the points in the subset.

We clearly remove all dependency from N, SoD computational requirements are $\mathcal{O}(m^3)$ time and $\mathcal{O}(m^2)$ space.

Learning iterations have the same time and space complexity given the following marginal likelihood

$$p(\mathbf{y}|\mathbf{X}_m) = -\frac{1}{2}\mathbf{y}^T(\mathbf{K}_{mm} + \sigma^2\mathbf{I})^{-1} - \frac{1}{2}\log|\mathbf{K}_{mm} + \sigma^2\mathbf{I}| - \frac{n}{2}\log(2\pi). \tag{44}$$

CVI model:

CVI maximize a lower bound (ELBO) on the marginal likelihood:

$$max_{\boldsymbol{\lambda}}\mathcal{L}(\boldsymbol{\lambda}) = \mathbb{E}_q[\log p(\mathbf{y}, \mathbf{z}) - \log q(\mathbf{z}|\boldsymbol{\lambda})], \tag{45}$$

where $\boldsymbol{\lambda}$ are parameters of the variational distribution $q(\mathbf{z}|\boldsymbol{\lambda})$ and $\mathbf{z}$ are any latent variables. In each iteration CVI execute the following update.

$$q_{t+1} \propto \left(e^{\langle\phi(\mathbf{z}),\mathbf{g}_t\rangle}p(\mathbf{z})\right)^{\beta_t} q_t^{1-\beta_t} \tag{46}$$

where β_t is the step size q_t is the posterior at step t, $\phi(\mathbf{z})$ are the sufficient statistics and $\mathbf{g}_t \triangleq \hat{\nabla}_\mu \mathbb{E}[\log p(\mathbf{y}|\mathbf{z})]|_{\mu=\mu_t}$, is the gradient of the expected likelihood with respect to the mean parameters.

CVI updates in GP regression is equivalent to:

$$q_t \propto \left[\prod_{i \in I_t} \mathcal{N}(y_i|z_i, 1/\tilde{\lambda}_{i,t})\right]\mathcal{N}(\mathbf{z}|0, \mathbf{K}_{tt}) \tag{47}$$

where I_t is the set of datapoints seen so far and $\mathbf{K}_{tt}$ is the covariance between points in I_t. $\tilde{\lambda}_{i,t} = (1 - r_t)\tilde{\lambda}_{i,t} + r_t\frac{N}{\sigma^2}$ for the index i selected at time t and $\tilde{\lambda_{i,t}} = (1 - r_t)\tilde{\lambda}_{i,t}$ for the rest, where $r_t = \frac{\beta_t}{(1+\beta_t)}$.

$$\mathcal{L}(\mathbf{m}, \mathbf{V}) = \sum_n E[\log p(y_n|z_n)] - D_{KL}[q(z|y)||\mathcal{N}(z|0, K)] \tag{48}$$

Initially all the precision are set to zero $\boldsymbol{\lambda} = 0\ 0$. Note that CVI allows each datapoint to have its own variance while SoD restrict all datapoints to have the same variance.

5 Hyperparameters Learning

5.1 Description

We compare three different learning methods: SoD, varDTC, SVI-GP. Details on SoD have already been given at the end of section 4.1.3.

varDTC and SVI-GP approximate the posterior distribution $p(f_*|\mathbf{y})$, using only a small amount of inducing variables $\mathbf{f}_u = f(X_u)$. Note that in order to learn the inducing points X_u we don't consider them as a subset of the training points, the only constraint we put on them is that lie in the same space than the data $\mathbf{X}$.

Both method approximate the exact posterior distribution

$$p(f_*|\mathbf{y}) = \int p(f_*|\mathbf{f}_u, \mathbf{f})p(\mathbf{f}|\mathbf{f}_u, \mathbf{y})p(\mathbf{f}_u|\mathbf{y})d\mathbf{f}\mathbf{f}_u \tag{49}$$

assuming that f_* is independent of $\mathbf{f}$ given $\mathbf{f}_u$.

$$q(\mathbf{z}) = \int p(\mathbf{z}|\mathbf{f}_u)p(\mathbf{f}_u|\mathbf{y})d\mathbf{f}_u. \tag{50}$$

Both methods instead of computing the the exact conditional distribution $p(\mathbf{f}_u|\mathbf{y})$, introduce a variational Gaussian distribution $\phi(\mathbf{f}_u) \approx p(\mathbf{f}_u|\mathbf{y})$ with mean $\boldsymbol{\mu}$ and variance $\mathbf{V}$.

varDTC compute the closed form expression of the parameters $\boldsymbol{\mu}$ and $\mathbf{V}$ that minimize the Kullback-Leiber divergence between the exact augmented posterior and the approximate augmented posterior $KL(q(\mathbf{f}, \mathbf{f}_u)||p(\mathbf{f}, \mathbf{f}_u|\mathbf{y}))$, where $q(\mathbf{f}, \mathbf{f}_u|\mathbf{y}) = p(\mathbf{f}|\mathbf{f}_u)\phi(\mathbf{f}_u)$.

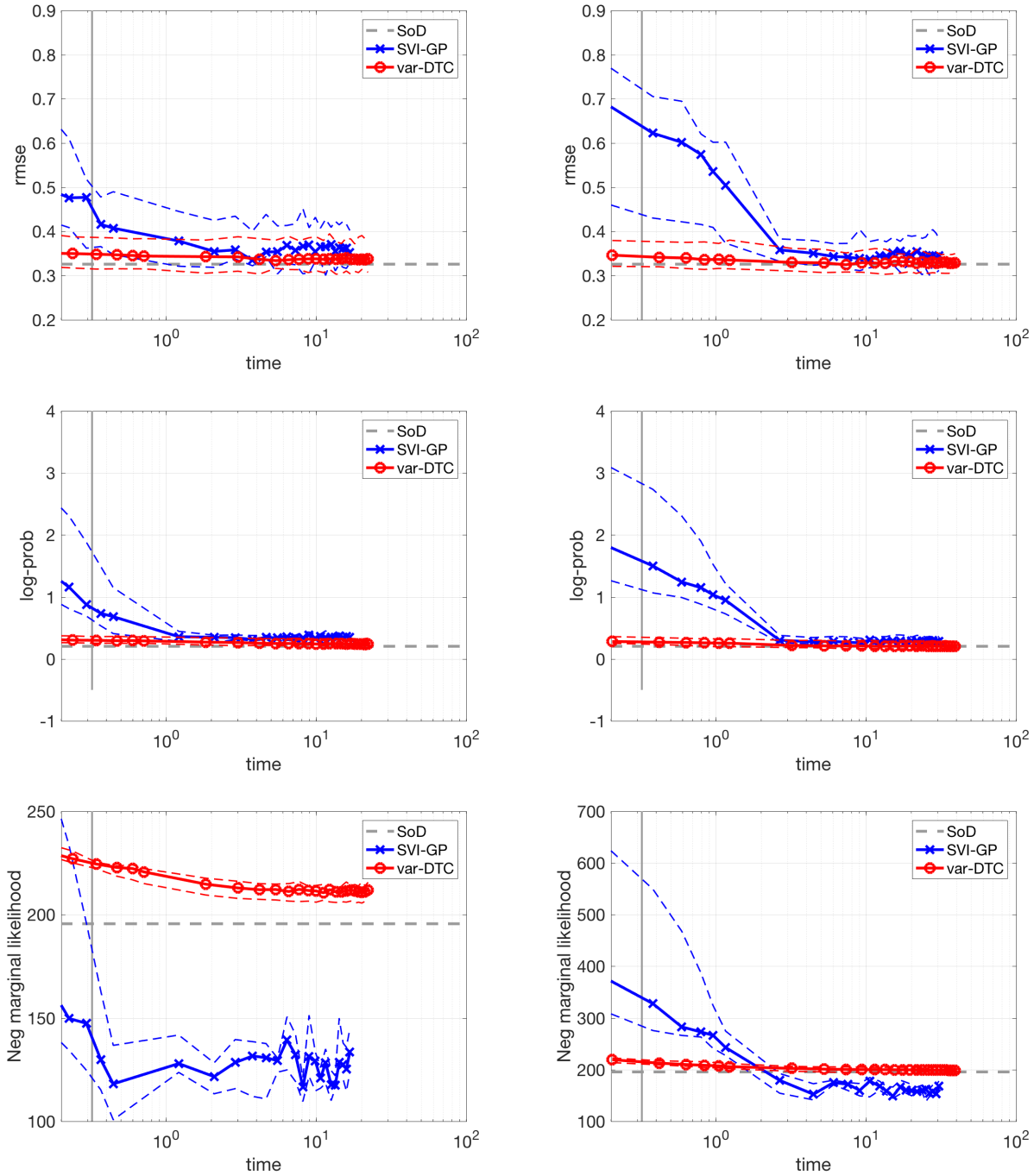
$$\begin{aligned} \boldsymbol{\mu}^* &= \sigma^{-2}\mathbf{K}_{uu}\boldsymbol{\Sigma}\mathbf{K}_{uf}\mathbf{y} \\ \mathbf{A}^* &= \mathbf{K}_{uu}\boldsymbol{\Sigma}\mathbf{K}_{uu} \end{aligned} \tag{51}$$

Instead of computing analytical expression for the two variational parameters SVI-GP, minimize the Kullback-Leiber divergence between the exact and approximate posterior using stochastic updates (more details on varDTC and SVI-GP are given in section 5.3).

We show results for Housing dataset in Figures 4 and 5, results for other datasets can be found in Appendix in Figures 21, 22, 23 and 24.

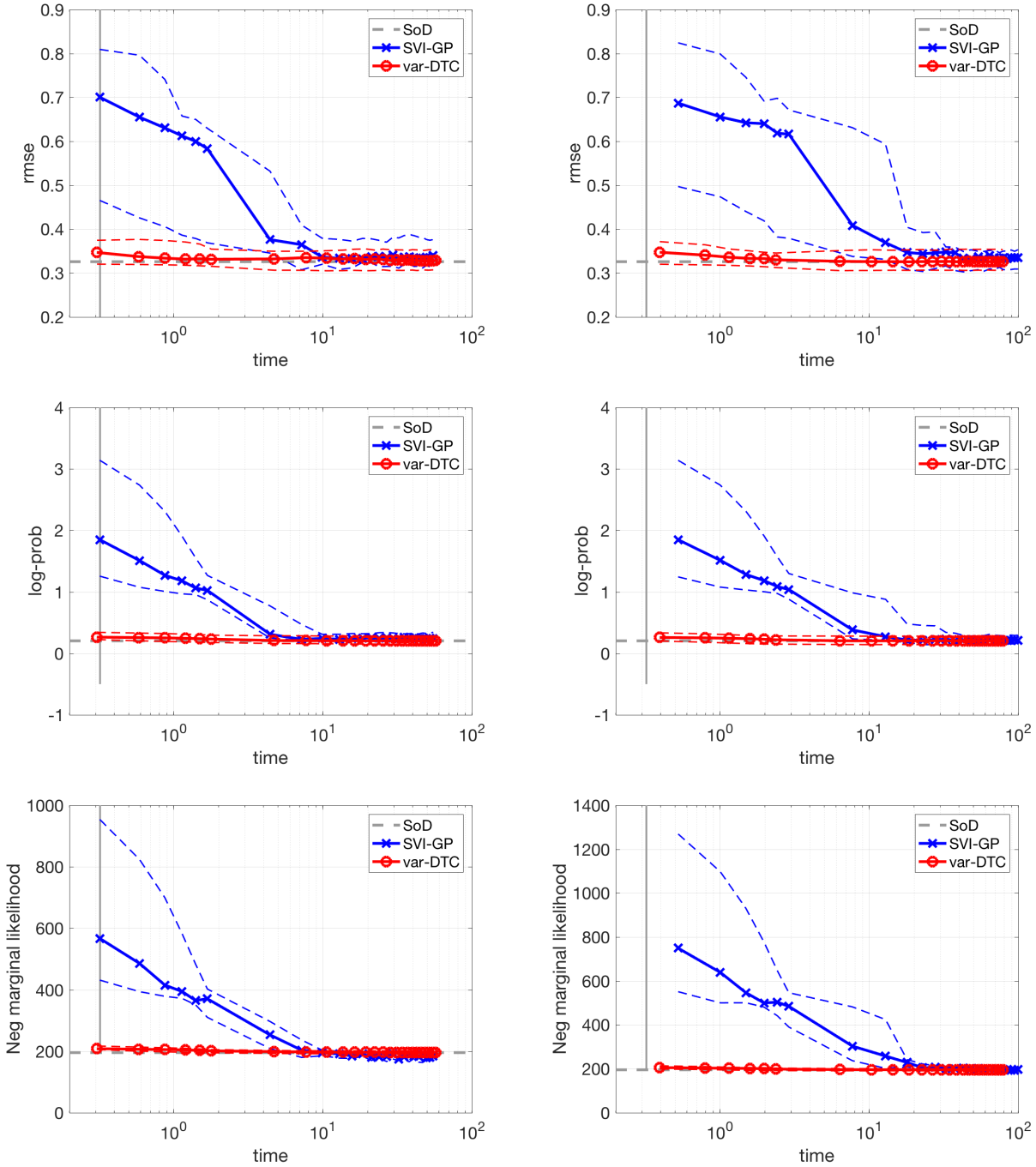
In order to compare these three different learning algorithms, we plot the three different loss function: rmse, mean log-probability and negative marginal likelihood as a function of the computation time spend by the method on learning. We repeat this experiment on 10 different seeds. Results show that SVI-GP is more sensitive to noise

5.2 Experimental Results



100 inducing points 200 inducing points

Figure 4: This figure shows results obtained on **Housing** dataset, each row of figures shows a different loss function. The left column shows results obtained using 100 inducing points and the right columns shows results obtained using 200 inducing points. The vertical grey line shows the time required to compute the SoD predictions using quasi-Newton optimization algorithm from *MinFunc* library. We see that varDTC needs much less steps than SVI-GP to converge. We also note improvement for both methods when increasing the number of inducing points.



300 inducing points

400 inducing points

Figure 5: This figure shows results obtained on **Housing** dataset, each row of figures shows a different loss function. The left column shows results obtained using 300 inducing points and the right columns shows results obtained using 400 inducing points. The vertical grey line shows the time required to compute the SoD predictions using quasi-Newton optimization algorithm from *MinFunc* library. We see that varDTC needs much less steps than SVI-GP to converge. We also note that SVI-GP is more sensitive to initialization.

5.3 Details

varDTC model:

varDTC approximate the posterior distribution $p(\mathbf{f}_*|\mathbf{y})$, using only a small amount of inducing variables $\mathbf{f}_u$. The exact posterior can be written as follow :

$$p(\mathbf{f}_*|\mathbf{y}) = \int p(\mathbf{f}_*|\mathbf{f}_u, \mathbf{f})p(\mathbf{f}|\mathbf{f}_u, \mathbf{y})p(\mathbf{f}_u|\mathbf{y})d\mathbf{f}\mathbf{f}_u \tag{52}$$

We assume that $\mathbf{f}_u$ are sufficient statistics for the parameter $\mathbf{f}$. It means that $\mathbf{f}_*$ and $\mathbf{f}$ are independent given $\mathbf{f}_u$, $p(\mathbf{f}_*|\mathbf{f}_u, \mathbf{f}) = p(\mathbf{f}_*|\mathbf{f}_u)$.

This leads to the following approximation of the posterior (see [3] equation (5)) :

$$q(\mathbf{f}_*) = \int p(\mathbf{f}_*|\mathbf{f}_u)p(\mathbf{f}_u|\mathbf{y})d\mathbf{f}_u \tag{53}$$

In general, $\mathbf{f}_u$ are not sufficient statistics, hence $q(\mathbf{f}_*)$ is an approximation of the true the posterior. We also approximate $p(\mathbf{f}_u|\mathbf{y})$ by $\phi(\mathbf{f}_u)$ a variational Gaussian distribution with mean $\boldsymbol{\mu}$ and covariance $\mathbf{A}$. The parameters of the predictive Gaussian distribution can be written as follow :

$$\bar{\mathbf{f}_*} = \mathbf{K}_{*u}\mathbf{K}_{uu}^{-1}\boldsymbol{\mu} \tag{54}$$

$$cov(\mathbf{f}_*) = k_{**} - \mathbf{K}_{*u}(\mathbf{K}_{uu}^{-1} + \mathbf{K}_{uu}^{-1}\mathbf{A}\mathbf{K}_{uu}^{-1})\mathbf{K}_{u*} \tag{55}$$

To select the parameters $\boldsymbol{\mu}$, $\mathbf{A}$ we minimize the Kullback-Leiber divergence between the true posterior $p(\mathbf{f}, \mathbf{f}_u|\mathbf{y})$ and the approximate posterior $q(\mathbf{f}, \mathbf{f}_u|\mathbf{y}) = p(\mathbf{f}|\mathbf{f}_u)\phi(\mathbf{f}_u)$. Minimizing $KL(q(\mathbf{f}, \mathbf{f}_u)||p(\mathbf{f}, \mathbf{f}_u|\mathbf{y}))$ is equivalent to maximizing the following bound (see [3] equation (8)) :

$$\mathcal{L}(\boldsymbol{\mu}, \mathbf{A}) = \int p(\mathbf{f}|\mathbf{f}_u)\phi(\mathbf{f}_u) \log \frac{p(\mathbf{y}|\mathbf{f})p(\mathbf{f}_u)}{\phi(\mathbf{f}_u)} d\mathbf{f}d\mathbf{f}_u \tag{56}$$

Taking derivate of the bound w.r.t to variational parameters $\boldsymbol{\mu}$, $\mathbf{A}$ and setting it to zero allows us to find the optimal expression for the two variational parameters :

$$\begin{aligned} \boldsymbol{\mu}^* &= \sigma^{-2}\mathbf{K}_{uu}\boldsymbol{\Sigma}\mathbf{K}_{uf}\mathbf{y} \\ \mathbf{A}^* &= \mathbf{K}_{uu}\boldsymbol{\Sigma}\mathbf{K}_{uu} \end{aligned} \tag{57}$$

where $\boldsymbol{\Sigma} = (\mathbf{K}_{uu} + \sigma^{-2}\mathbf{K}_{uf}\mathbf{K}_{fu})^{-1}$.

The final predictive distribution is obtained by plugging the optimal parameters $\boldsymbol{\mu}^*$ and $\mathbf{A}^*$ found in eq. (57) into the predictive sparse GP distribution eq. (54).

We optimize inducing points and hyperparameters using the bound obtained after insertion of the optimal parameters eq. (57) into the bound eq. (56) :

$$\mathcal{L}(\mathbf{X}_u, \boldsymbol{\theta}) = \log[\mathcal{N}(0, \sigma^2\mathbf{I} + \mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{K}_{uf})] - \frac{1}{2\sigma^2}\mathrm{Tr}(\tilde{\mathbf{K}}) \tag{58}$$

where $\tilde{\mathbf{K}} = \mathbf{K}_{ff} - \mathbf{K}_{fu}\mathbf{K}_{uu}\mathbf{K}_{uf}$ and $\boldsymbol{\theta}$ are model hyperparameters.

Note that in practice when simultaneously learning the hyperparameters and the inducing points we need two different step sizes, which prevent us from using *minFunc* optimization library.

SVI-GP model:

In order to apply Stochastic Variational Inference on Gaussian Process Regression, we optimize the following bound (see [4] eq.(4)):

$$\mathcal{L} = \sum_{i=1}^{n} \left(\log \mathcal{N}(y_i | \mathbf{k}_i^T \mathbf{K}_{mm}^{-1} \mathbf{m}, \tilde{\mathbf{\Sigma}}) - \frac{1}{2}\tilde{\sigma}_{i,i}\tilde{k}_{i,i} - \frac{1}{2}\mathrm{Tr}(\mathbf{S}\mathbf{\Lambda}_i) \right) - KL(q(\mathbf{u})||p(\mathbf{u})) \tag{59}$$

where $\tilde{\mathbf{\Sigma}} = \sigma^{-2}\mathbf{I}$ with $\mathbf{I}$ the $N \times N$ identity matrix, $\tilde{\sigma}_{i,i}$ is the ith diagonal element of $\tilde{\mathbf{\Sigma}}$. $\tilde{k}_{i,i}$ is the ith diagonal element of $\tilde{\mathbf{K}}$, where $\tilde{\mathbf{K}} = \mathbf{K}_{ff} - \mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{K}_{uf}$. $\mathbf{\Lambda}_i = \sigma^{-2}\mathbf{K}_{uu}^{-1}\mathbf{k}_i\mathbf{k}_i^T\mathbf{K}_{uu}^{-1}$
And finally $\mathbf{m}$ and $\mathbf{S}$ are respectively the mean and the covariance parameters of the variational distribution.

In order to optimize variational parameters in $\mathcal{L}$, SVI takes step in direction of the natural gradient, which correspond to the classical gradients re-scaled by the inverse Fisher, matrix. Note that due to exponential families properties natural gradients of $\mathcal{L}$ are equivalent to gradients taken with respect to the mean parameters, η (see [4]). In the experiments in order to learn the hyperparameters θ as well as the variational parameters $\boldsymbol{\mu}$ and $\mathbf{A}$ we used implicit gradient algorithms described below :

Algorithm 2 Implicit gradients

input: $\mathcal{L}(\theta_1, \theta_2)$.
for s=1...S:
  for t=1...T:
    $\lambda_2^{t+1} = \lambda_2^t + r_1 \frac{\partial \mathcal{L}(\lambda_1^s, \lambda_2^t)}{\partial \eta_2^t}$.
  goto *inner loop.*
    $\lambda_1^{s+1} = \lambda_1^{s+1} + r_2 \frac{\partial \mathcal{L}(\lambda_1^s, \lambda_2^*)}{\partial \eta_1^t}$.
goto *outer loop.*

Where r_1 and r_2 are two different step sizes.

The difference between the two methods, varDTC and SVI-GP is that while varDTC computes closed form solution for the variational hyperparameters SVI-GP optimize those parameters by taking stochastic gradient steps.

6 Conclusion

In this thesis we compared three different aprroximation algorithms that use a low-rank approximation of the GP kernel: Nyström approximation, Subset of Regressors method and Deterministic Training Conditional method. We showed that for posterior inference task DTC consistently performed better, and we discussed theoretical reason behind this performance.

We compared two different subsets of data methods: Subset of Data methode and Conjugate Variaional Inference algorithm. We show that CVI consistently performs better than SoD.

Using the results found in the comparison of approximate GP, we compared two methods, which learn hypererparameters and inducing point using variational learning. We showed that the stochastic approach (SVI) is more sensitive to initialization and gradient noise, however both methods converge with a reasonable number of inducing points.

For futur work, we should look at the performance of CVI algorithm for hyperparameter learning task. Since CVI consistently outperformed SoD in posterior inference task, it would be intersting to evaluate its performance for hyperparameters learning and compare it with the two variational method presented in this thesis.

References

[1] C. E. Rasmussen, and C. K. I. Williams *Gaussian Processes for Machine Learning* the MIT press, 2006

[2] Joaquin Quinonero-Candela and Carl Edward Rasmussen *A Unifying View of Sparse Approximate Gaussian Process Regression*

[3] Michalis K. Titsias *Variational Learning of Inducing Variables in Sparse Gaussian Processes*

[4] James Hensman, Nicolo Fusi and Neil D. Lawrence *Gaussian Processes for Big Data*

[5] Matthew D. Hoffman, David M. Blei, Chong Wang, John Paisley *Stochastic Variational Inference* Journal of Machine Learning Research 14 (2013) 1303-1347

[6] Christopher K. I. Williams, and Matthias Seeger. *Using the Nyström Method to Speed UP Kernel Machines.*

[7] Sanjiv Kumar, Mehryar Mohri, and Ameet Talwalkar. *Sampling methods for the Nyström Method.* Journal of Machine Learning Research 13 (2012) 981-1006

[8] Christopher K. I. Williams and Carl Edward Rasmussen and Anton Schwaighofer and Volker Tresp. Observations on the Nyström Method for Gaussian Processes.

[9] Mohammad E. Khan and Wu Lin. *Conjugate-Computation Variational Inference : Converting Variational Inference in Non-Conjugate Models to Inferences in Conjugate Models*

[10] Silverman, B. W. *Some Aspects of the Spline Smoothing Approach to Non-parametric Regression Curve Fitting (with discussion)*

[11] Matthias Seeger, Christopher K. I. Williams, and Neil Lawrence. *Fast forward selection to speed up sparse Gaussian process regression.*

[12] Lichman, M. *UCI Machine Learning Repository [http://archive.ics.uci.edu/ml].* Irvine, CA: University of California, School of Information and Computer Science (2013).

[13] I-Cheng Yeh, *Modeling of strength of high performance concrete using artificial neural networks.* Cement and Concrete Research, Vol. 28, No. 12, pp. 1797-1808 (1998).

7 Annexes

7.1 Gaussian Identities

If $\mathbf{x}$ and $\mathbf{y}$ are two jointly Gaussian random variables.

$$\begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu}_x\\ \boldsymbol{\mu}_y\end{bmatrix}, \begin{bmatrix}\mathbf{A} & \mathbf{C}\\ \mathbf{C}^T & \mathbf{B}\end{bmatrix}\right) \tag{60}$$

then the conditional distribution of $\mathbf{x}$ given $\mathbf{y}$ is

$$\mathbf{x}|\mathbf{y} \sim \mathcal{N}(\boldsymbol{\mu}_x + \mathbf{C}\mathbf{B}^{-1}(\mathbf{y} - \boldsymbol{\mu}_y), \mathbf{A} - \mathbf{C}\mathbf{B}^{-1}\mathbf{C}^T) \tag{61}$$

7.1.1 Trace trick

$$\log p(\mathbf{X}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{2}\log|\boldsymbol{\Sigma}| - \frac{1}{2}\mathbf{x}^T\boldsymbol{\Sigma}^{-1}\mathbf{x} + cons \tag{62}$$

$$= \frac{1}{2}\log|\boldsymbol{\Sigma}| - \frac{1}{2}\mathrm{Tr}(\boldsymbol{\Sigma}^{-1}\mathbf{x}\mathbf{x}^T) + const \tag{63}$$

7.1.2 Kullback-Leiber Divergence for Multivariate Gaussian

$$D_{KL}(\mathcal{N}_0||\mathcal{N}_1) = \frac{1}{2}\left(\mathrm{Tr}(\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\Sigma}_0) + (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)^T\boldsymbol{\Sigma}_1^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0) - k + \ln\frac{|\boldsymbol{\Sigma}_1|}{|\boldsymbol{\Sigma}_0|}\right) \tag{64}$$

7.2 Matrix operation

7.2.1 Matrix derivatives

$$\begin{aligned}\frac{\partial(\mathbf{X}\mathbf{Y})}{\partial\Theta} &= \mathbf{X}\frac{\partial(\mathbf{Y})}{\partial\Theta} + \frac{\partial(\mathbf{X})}{\partial\Theta}\mathbf{Y}\\ \frac{\partial\mathbf{K}^{-1}}{\partial\Theta} &= -\mathbf{K}^{-1}\frac{\partial\mathbf{K}}{\partial\Theta}\mathbf{K}^{-1}\end{aligned} \tag{65}$$

7.2.2 Matrix Inversion Lemma

$$(\mathbf{A} + \mathbf{U}\mathbf{C}\mathbf{V})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}(\mathbf{C}^{-1} + \mathbf{V}\mathbf{A}^{-1}\mathbf{U})^{-1}\mathbf{V}\mathbf{A}^{-1} \tag{66}$$

where $\mathbf{A}$ is $n \times n$, $\mathbf{U}$ is $n \times k$, $\mathbf{V}$ is $k \times n$ and $\mathbf{C}$ is $k \times k$

7.2.3 Matrix Determinant Lemma

Given a $m \times m$ invertible matrix $\mathbf{W}$ two $n \times m$ matrices, $\mathbf{U}$ and $\mathbf{V}$, and a diagonal matrix $\mathbf{A}$.

$$\det(\mathbf{A} + \mathbf{U}\mathbf{W}\mathbf{V}^T) = \det(\mathbf{W}^{-1} + \mathbf{V}^T\mathbf{A}^{-1}\mathbf{U})\det(\mathbf{W})\det(\mathbf{A}). \tag{67}$$

8 Appendix

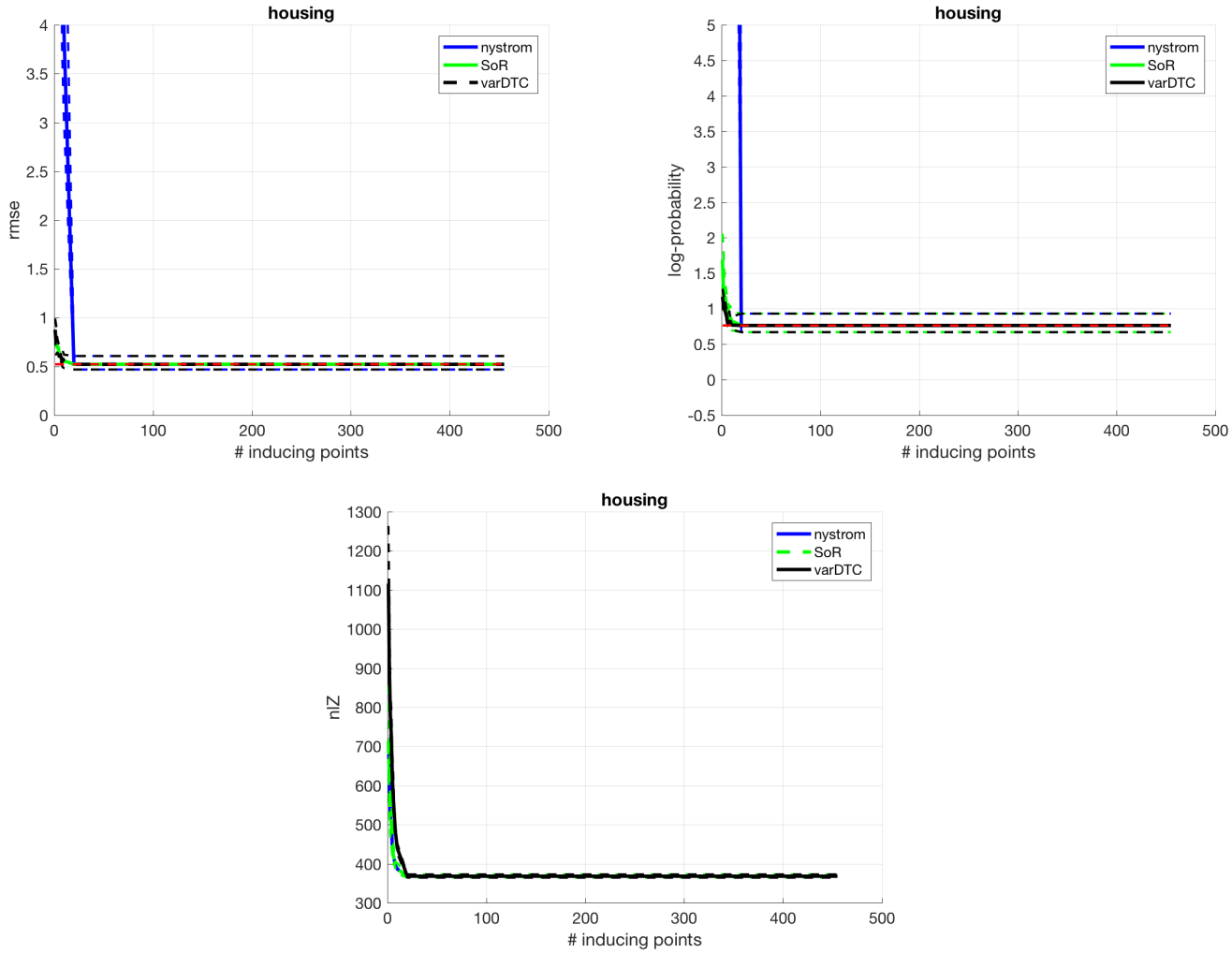


Figure 6: This figure shows the performance of the three low-rank approximation kernel algorithms as a function of the number of inducing points on **Housing** dataset. In this experiment the kernel used is the **LINiso** kernel. Each figure shows a different loss function. The red dotted line shows the performance of exact GP regression. Dashed line represent the 25th and 75th percentile while solid lines shows the median.

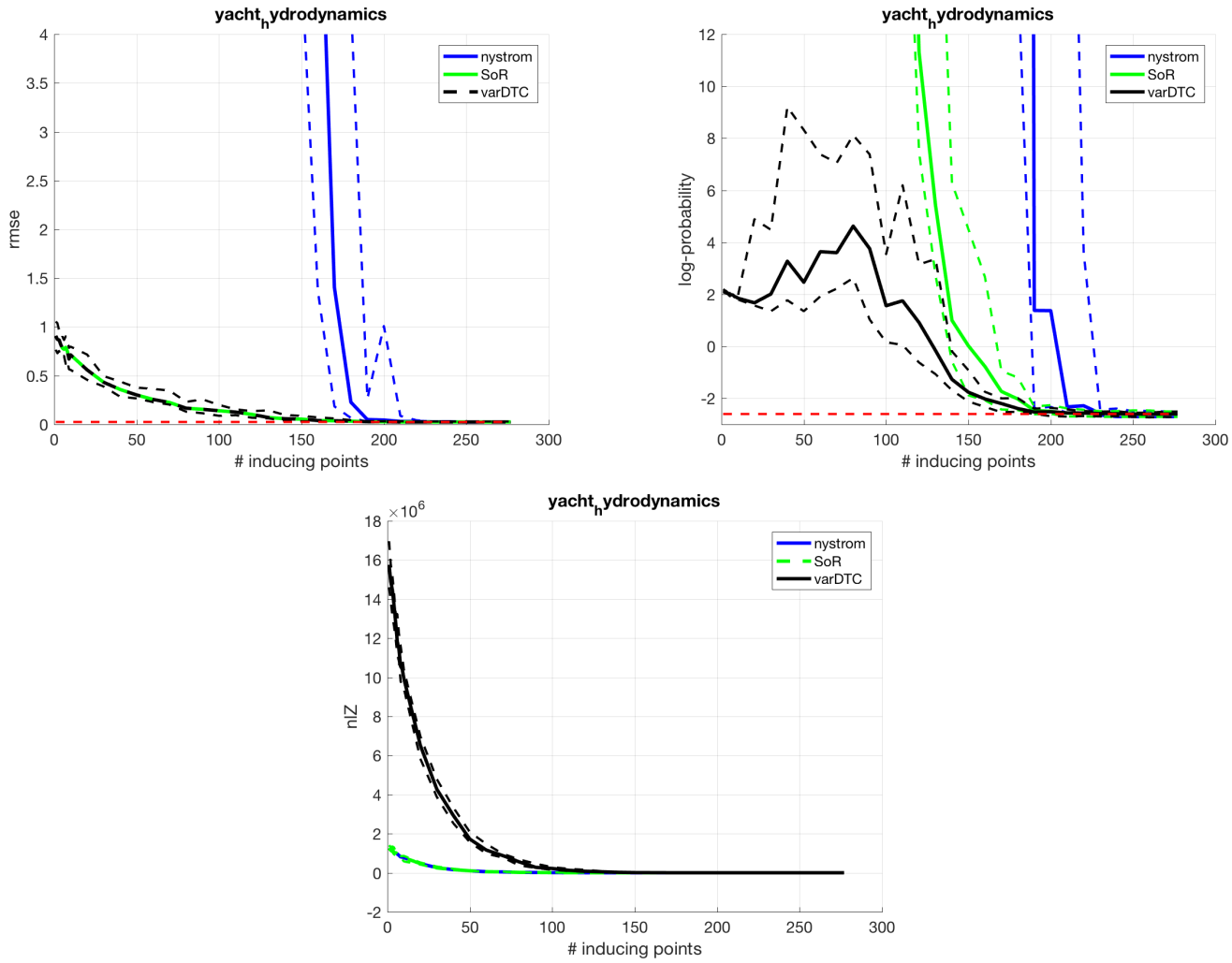


Figure 7: This figure shows the performance of the three low-rank approximation kernel algorithms as a function of the number of inducing points on **Yacht hydrodynamics** dataset. In this experiment the kernel used is the **SEiso** kernel. Each figure shows a different loss function. The red dotted line shows the performance of exact GP regression. Dashed line represent the 25th and 75th percentile while solid lines shows the median.

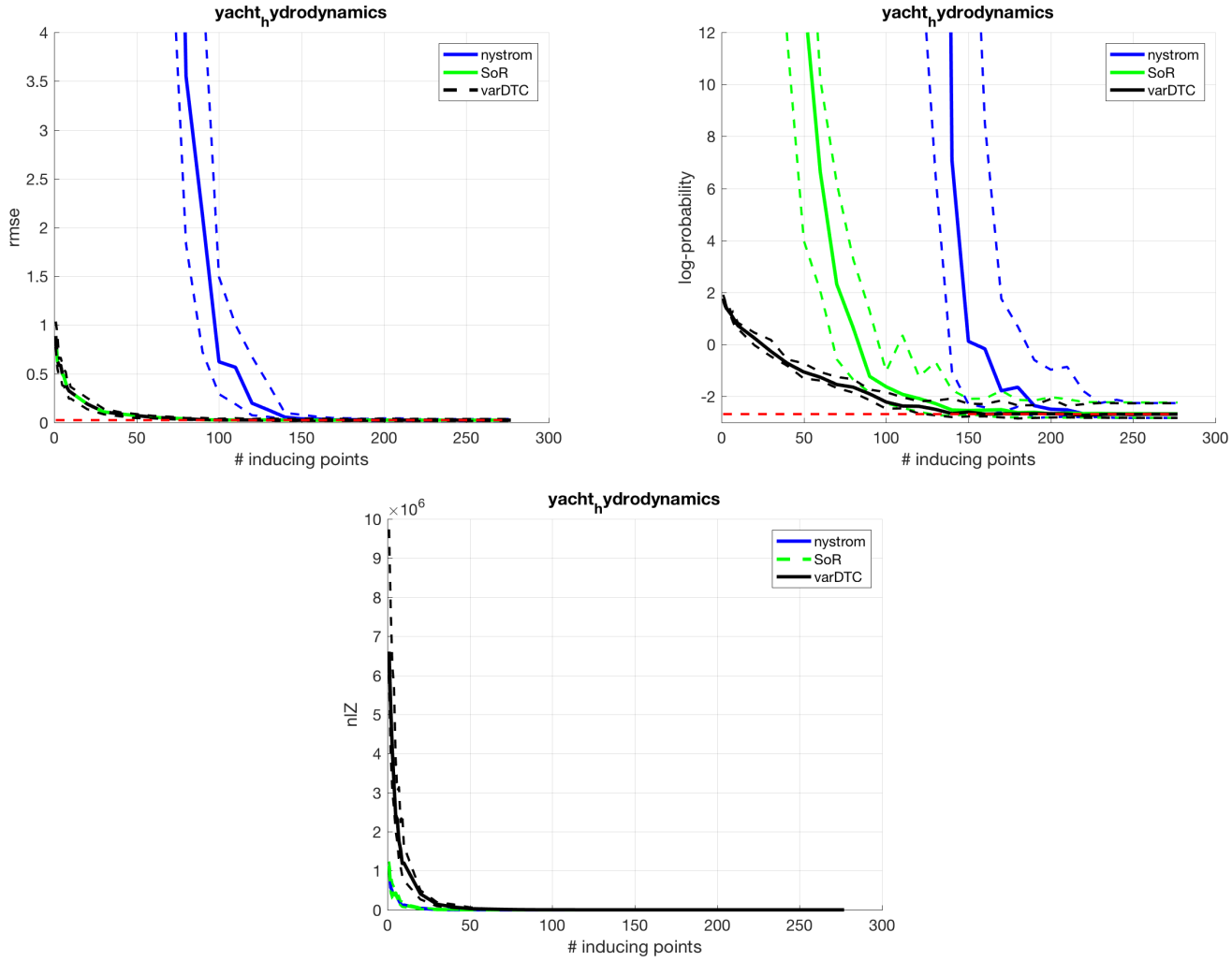


Figure 8: This figure shows the performance of the three low-rank approximation kernel algorithms as a function of the number of inducing points on **Yacht hydrodynamics** dataset. In this experiment the kernel used is the **SEard** kernel. Each figure shows a different loss function. The red dotted line shows the performance of exact GP regression. Dashed line represent the 25th and 75th percentile while solid lines shows the median.

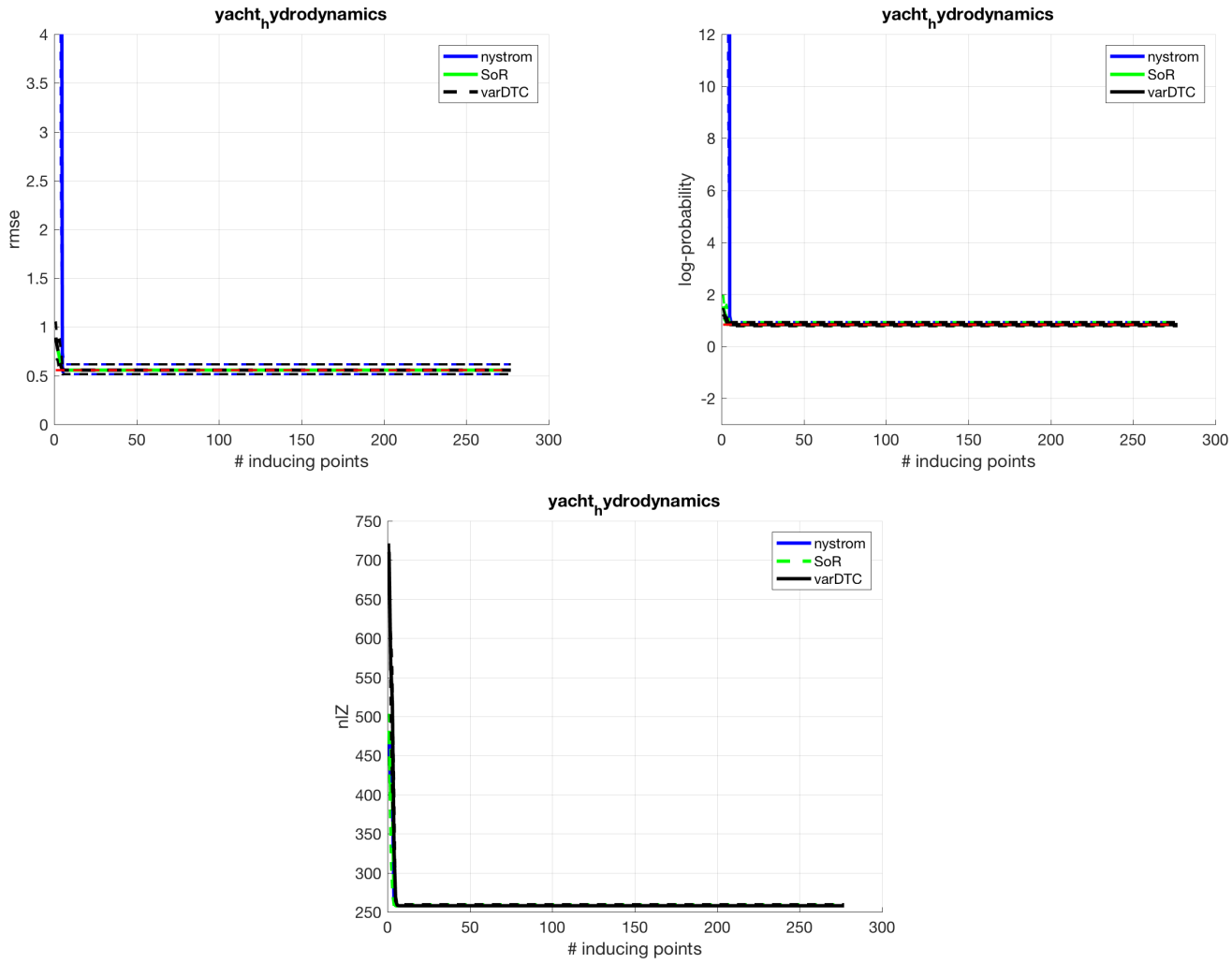


Figure 9: This figure shows the performance of the three low-rank approximation kernel algorithms as a function of the number of inducing points on **Yacht hydrodynamics** dataset. In this experiment the kernel used is the **LINiso** kernel. Each figure shows a different loss function. The red dotted line shows the performance of exact GP regression. Dashed line represent the 25th and 75th percentile while solid lines shows the median. Linear kernels capture only very simple characteristics of the data, hence the are not difficult to approximate and leads to poorer performance than Square Exponential kernels.

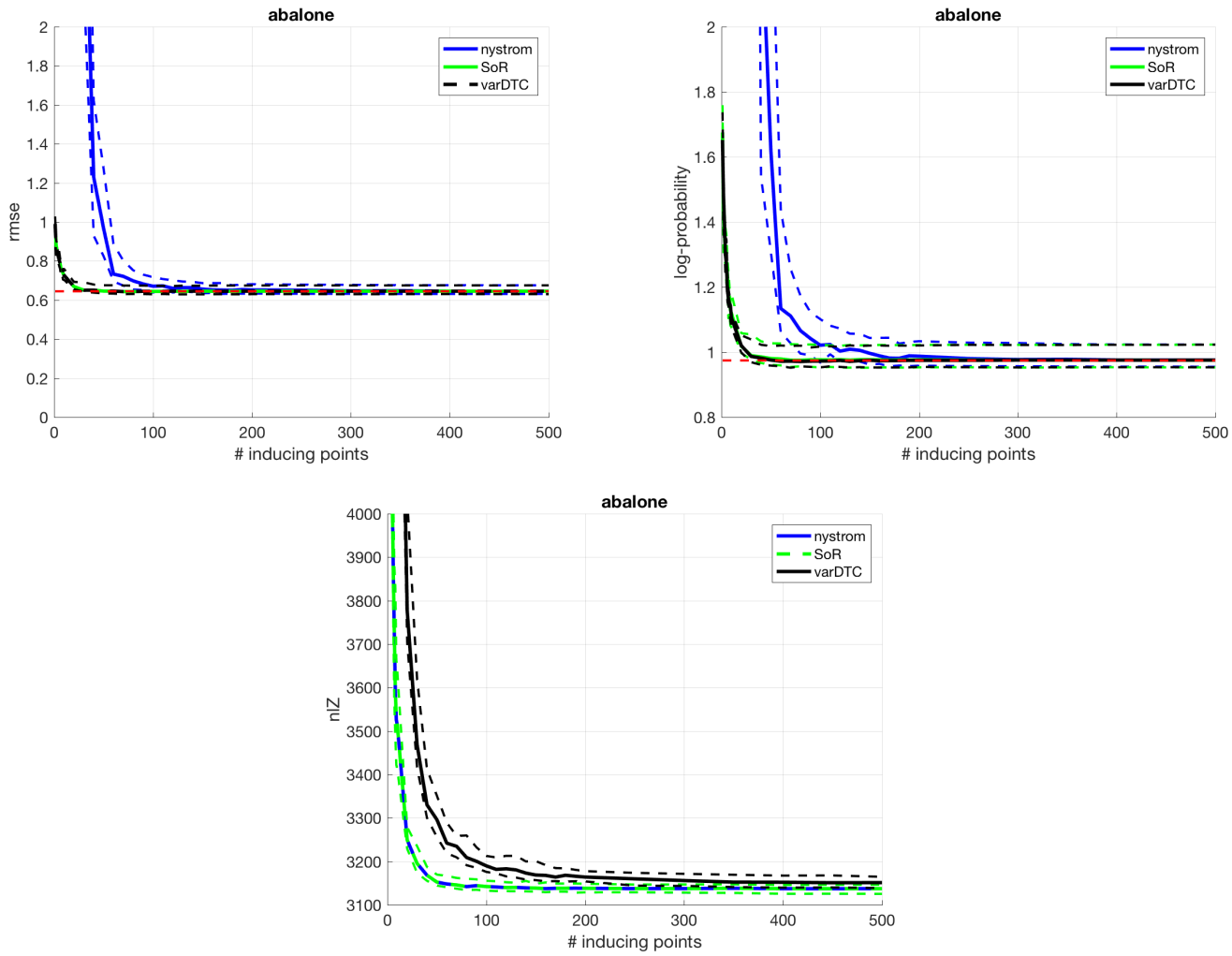


Figure 10: This figure shows the performance of the three low-rank approximation kernel algorithms as a function of the number of inducing points on **Abalone** dataset. In this experiment the kernel used is the **SEiso** kernel. Each figure shows a different loss function. The red dotted line shows the performance of exact GP regression. Dashed line represent the 25th and 75th percentile while solid lines shows the median. We see that rmse curves of DTC and SoR are exactly the same, but DTC's mean log-probability curve is performing better than SoR and Nyström curves. Note that when Nyström curves are extremely bad we don't even include them in the plot.

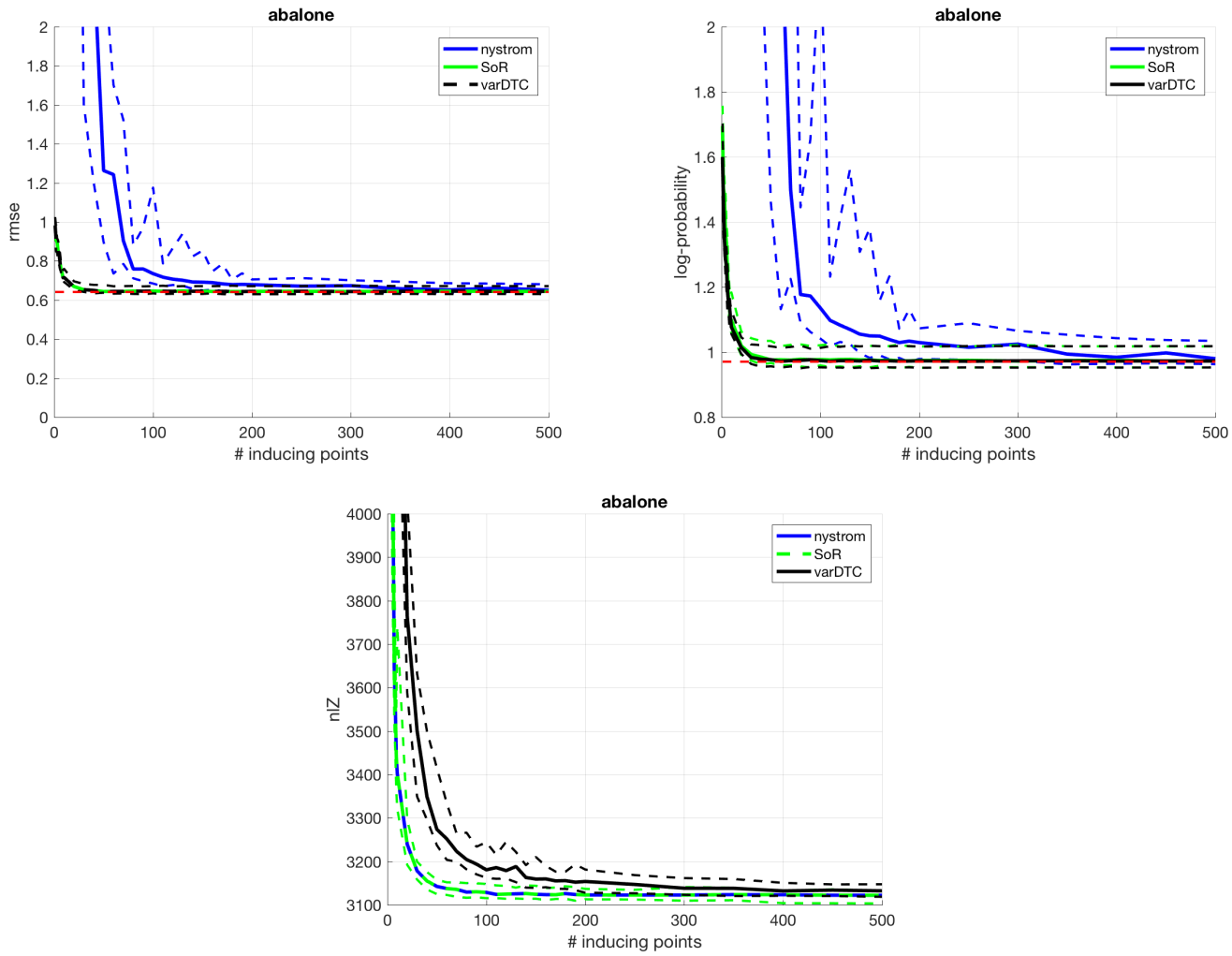


Figure 11: This figure shows the performance of the three low-rank approximation kernel algorithms as a function of the number of inducing points on **Abalone** dataset. In this experiment the kernel used is the **SEard** kernel. Each figure shows a different loss function. The red dotted line shows the performance of exact GP regression. Dashed line represent the 25th and 75th percentile while solid lines shows the median. We see that rmse curves of DTC and SoR are exactly the same, but DTC's mean log-probability curve is performing better than SoR and Nyström curves. Note that when Nyström curves are extremely bad we don't even include them in the plot.

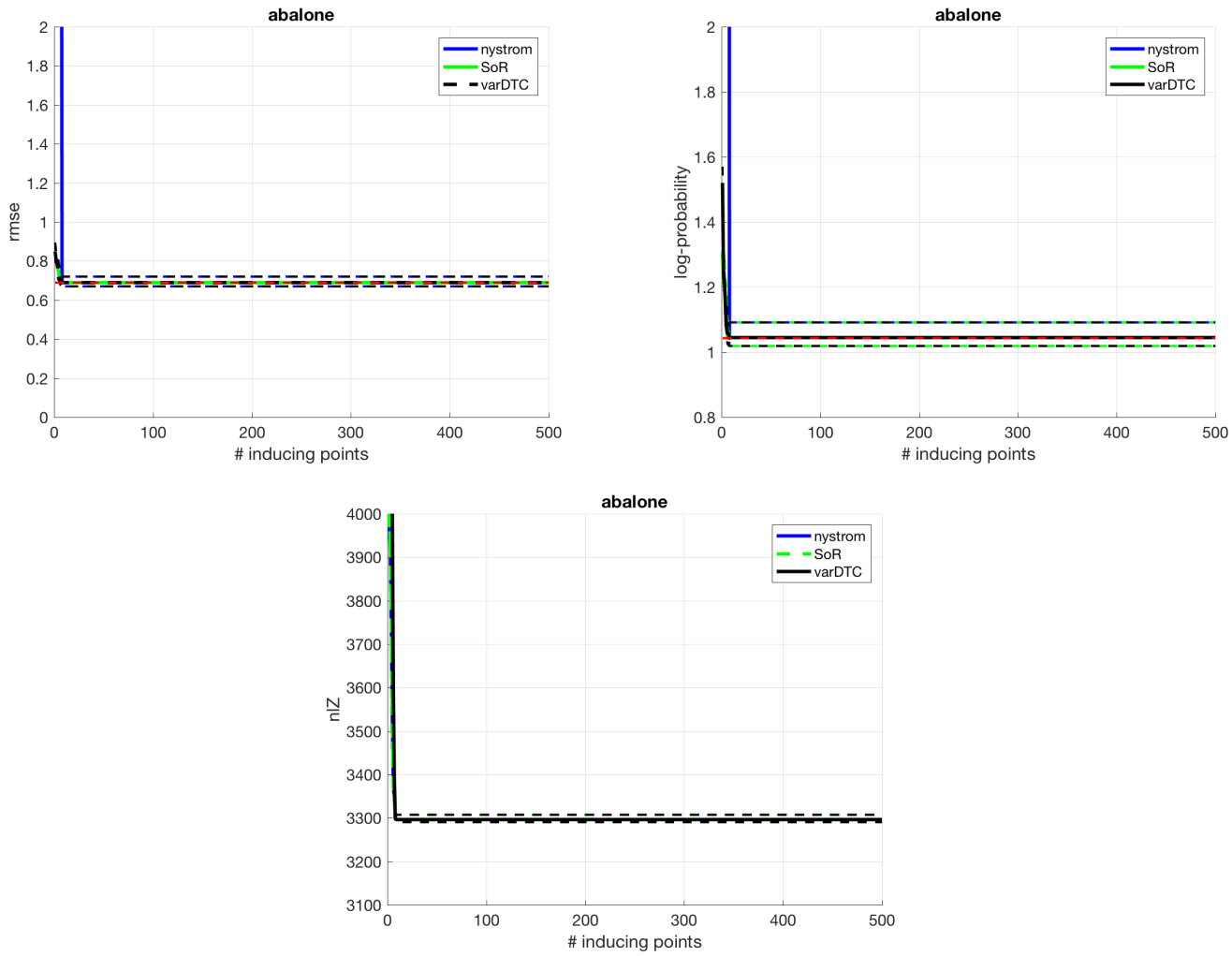


Figure 12: This figure shows the performance of the three low-rank approximation kernel algorithms as a function of the number of inducing points on **Abalone** dataset. In this experiment the kernel used is the **LINiso** kernel. Each figure shows a different loss function. The red dotted line shows the performance of exact GP regression. Dashed line represent the 25th and 75th percentile while solid lines shows the median. Linear kernels capture only very simple characteristics of the data, hence the are not difficult to approximate and leads to poorer performance than Square Exponential kernels.

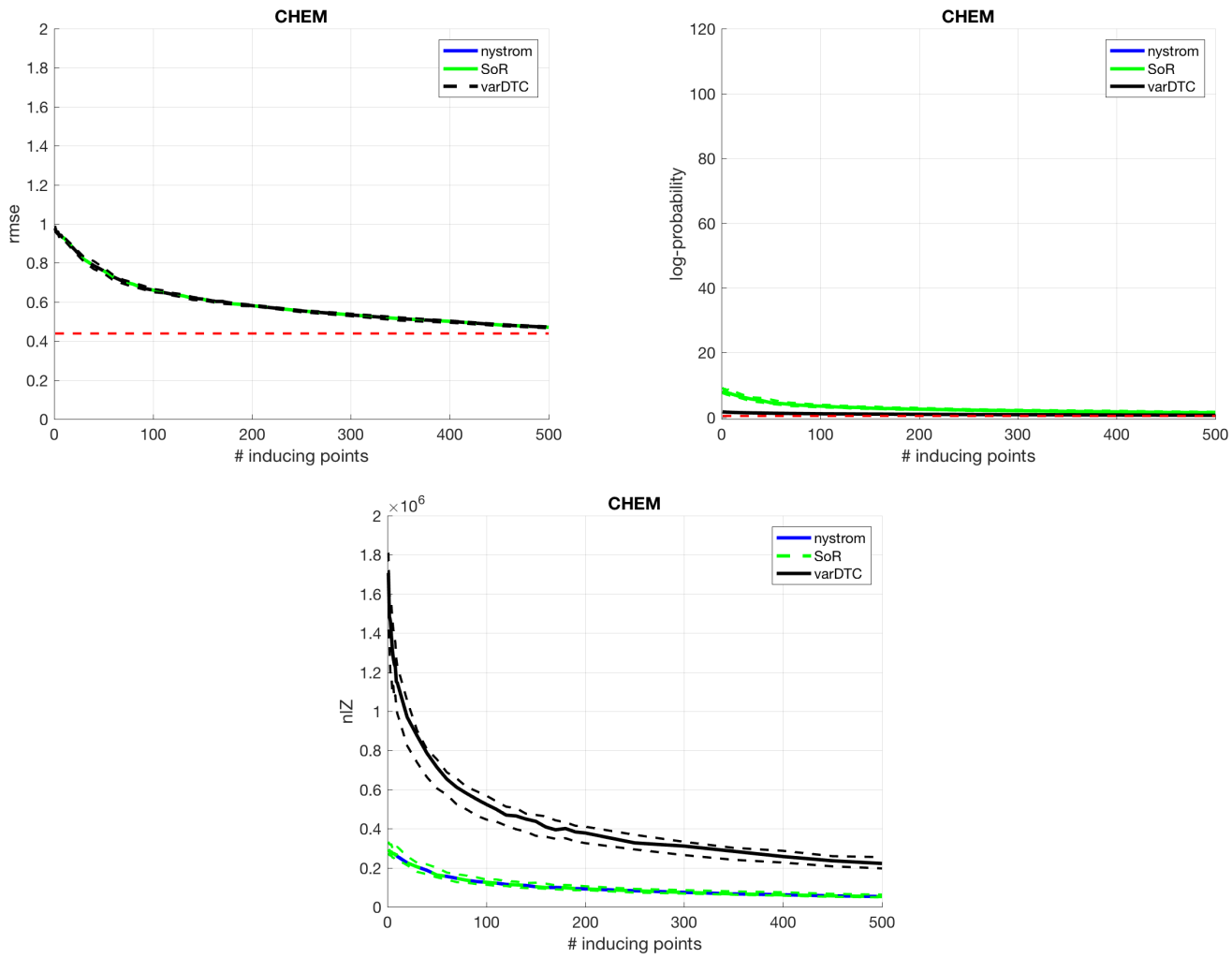


Figure 13: This figure shows the performance of the three low-rank approximation kernel algorithms as a function of the number of inducing points on **CHEM** dataset. In this experiment the kernel used is the **SEiso** kernel. Each figure shows a different loss function. The red dotted line shows the performance of exact GP regression. Dashed line represent the 25th and 75th percentile while solid lines shows the median. We see that rmse curves of DTC and SoR are exactly the same, but DTC's mean log-probability curve is performing better than SoR and Nyström curves. Note that when Nyström curves are extremely bad we don't even include them in the plot.

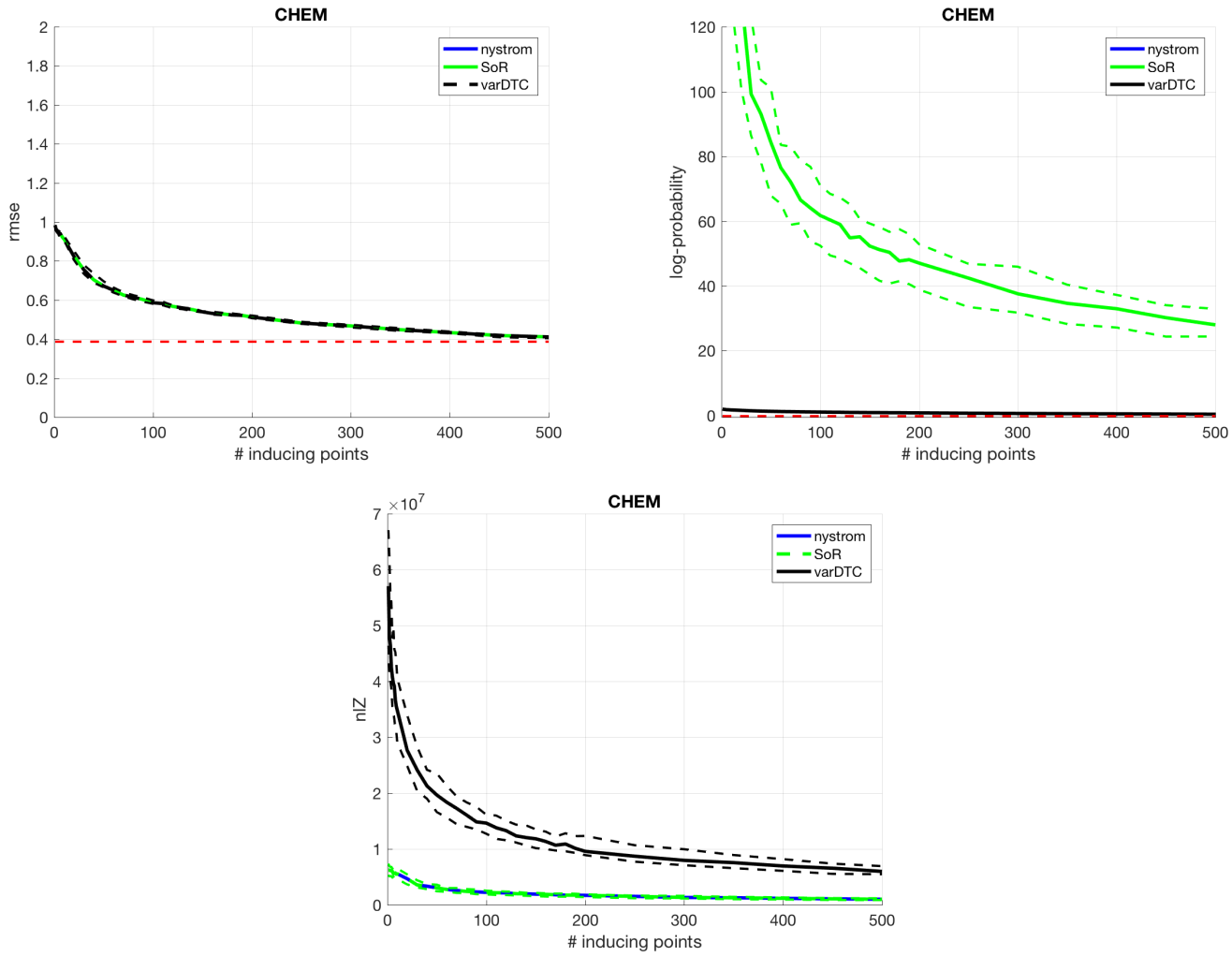


Figure 14: This figure shows the performance of the three low-rank approximation kernel algorithms as a function of the number of inducing points on **CHEM** dataset. In this experiment the kernel used is the **SEard** kernel. Each figure shows a different loss function. The red dotted line shows the performance of exact GP regression. Dashed line represent the 25th and 75th percentile while solid lines shows the median. We see that rmse curves of DTC and SoR are exactly the same, but DTC's mean log-probability curve is performing better than SoR and Nyström curves. Note that when Nyström curves are extremely bad we don't even include them in the plot.

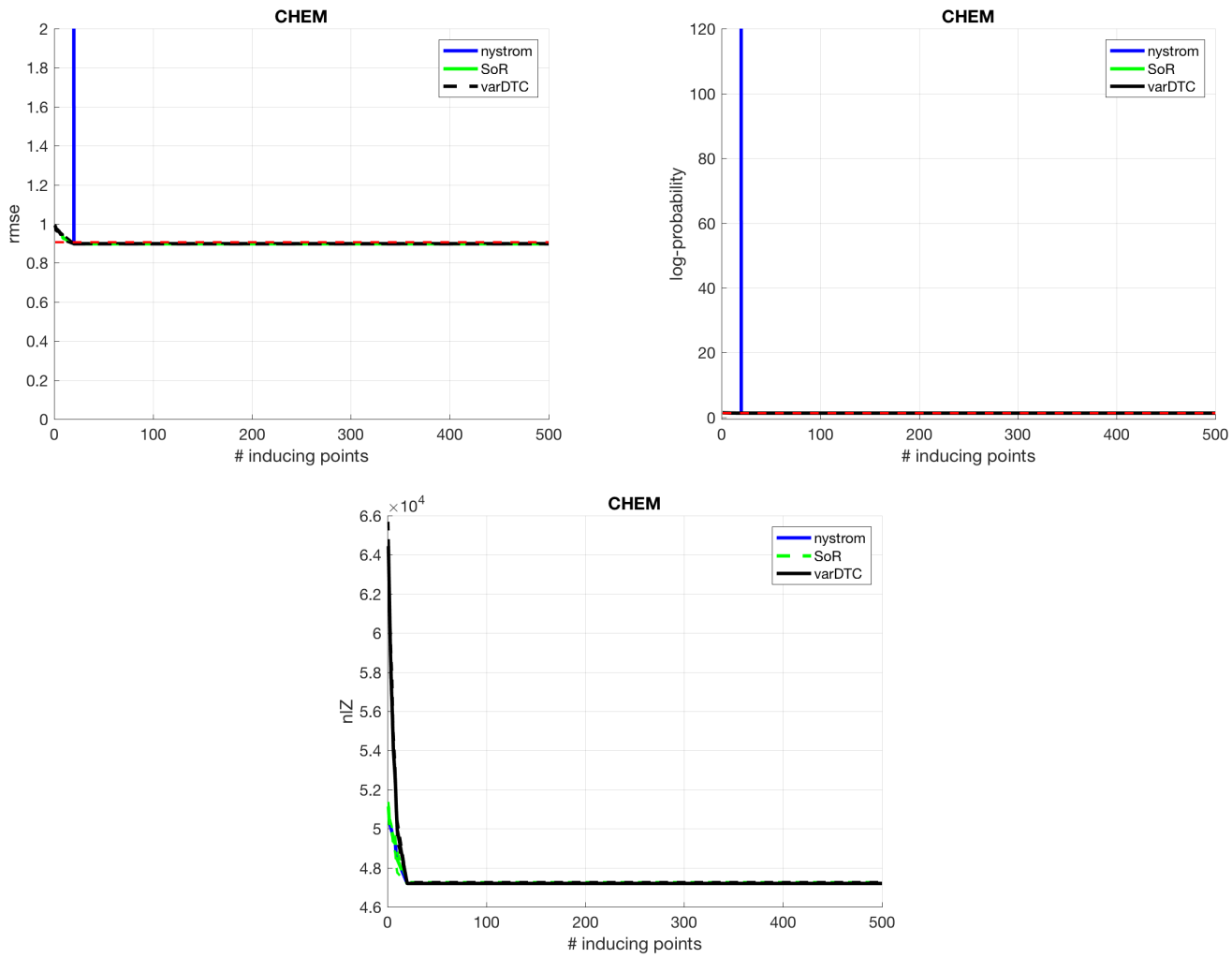


Figure 15: This figure shows the performance of the three low-rank approximation kernel algorithms as a function of the number of inducing points on **CHEM** dataset. In this experiment the kernel used is the **LINiso** kernel. Each figure shows a different loss function. The red dotted line shows the performance of exact GP regression. Dashed line represent the 25th and 75th percentile while solid lines shows the median. Linear kernels capture only very simple characteristics of the data, hence the are not difficult to approximate and leads to poorer performance than Square Exponential kernels.

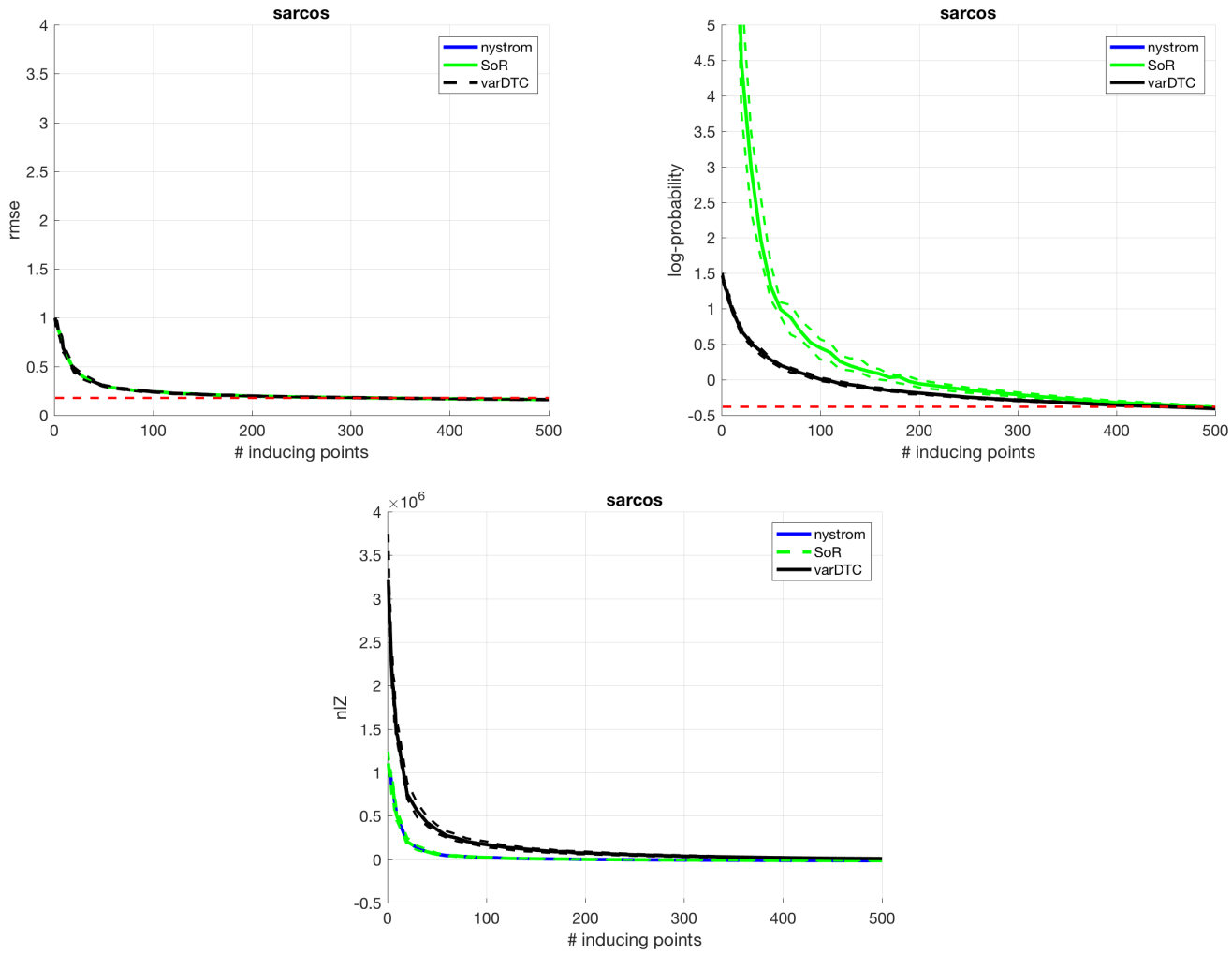


Figure 16: This figure shows the performance of the three low-rank approximation kernel algorithms as a function of the number of inducing points on **Sarcos** dataset. In this experiment the kernel used is the **SEiso** kernel. Each figure shows a different loss function. The red dotted line shows the performance of exact GP regression. Dashed line represent the 25th and 75th percentile while solid lines shows the median. We see that rmse curves of DTC and SoR are exactly the same, but DTC's mean log-probability curve is performing better than SoR and Nyström curves. Note that when Nyström curves are extremely bad we don't even include them in the plot.

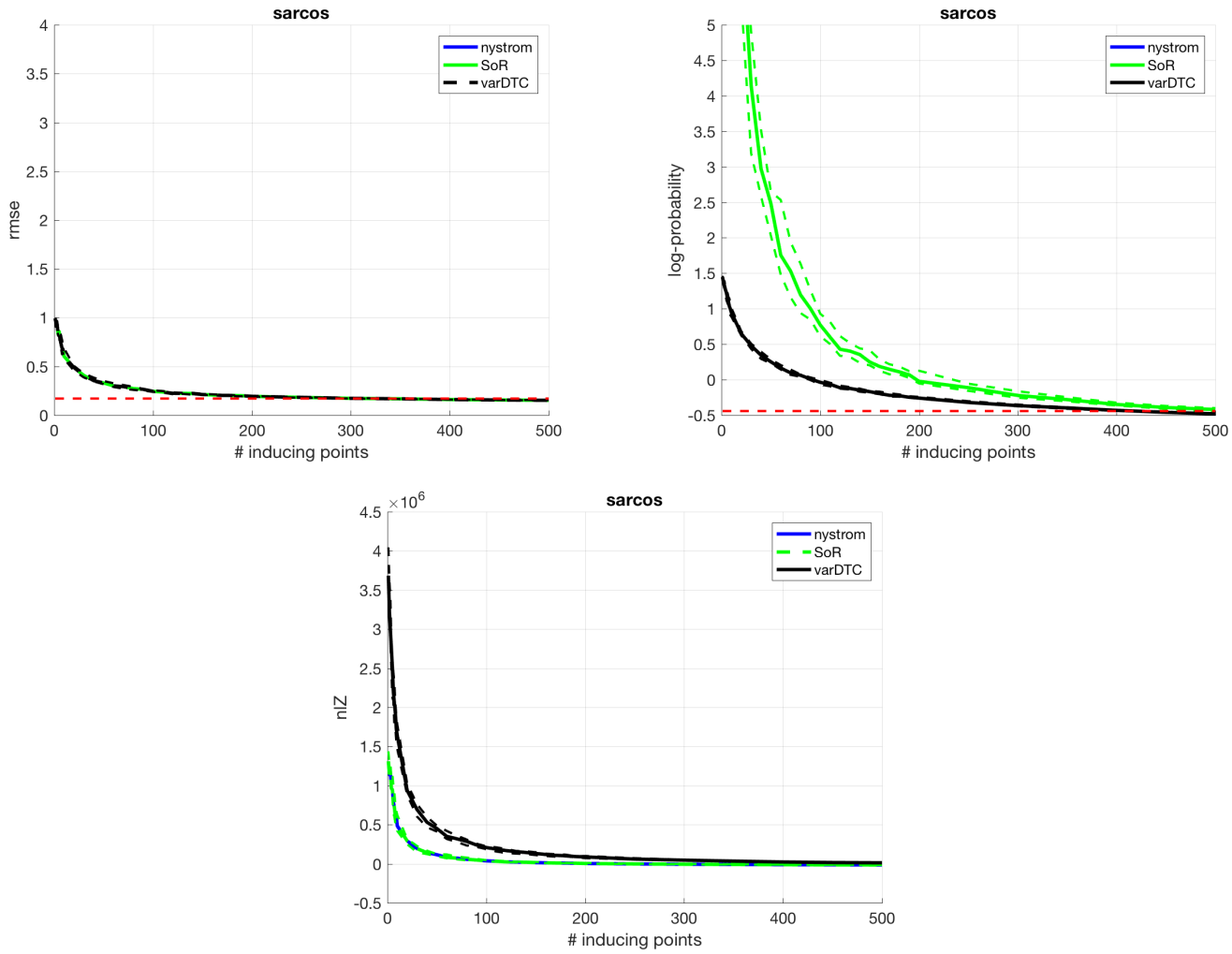


Figure 17: This figure shows the performance of the three low-rank approximation kernel algorithms as a function of the number of inducing points on **Sarcos** dataset. In this experiment the kernel used is the **SEard** kernel. Each figure shows a different loss function. The red dotted line shows the performance of exact GP regression. Dashed line represent the 25th and 75th percentile while solid lines shows the median. We see that rmse curves of DTC and SoR are exactly the same, but DTC's mean log-probability curve is performing better than SoR and Nyström curves. Note that when Nyström curves are extremely bad we don't even include them in the plot.

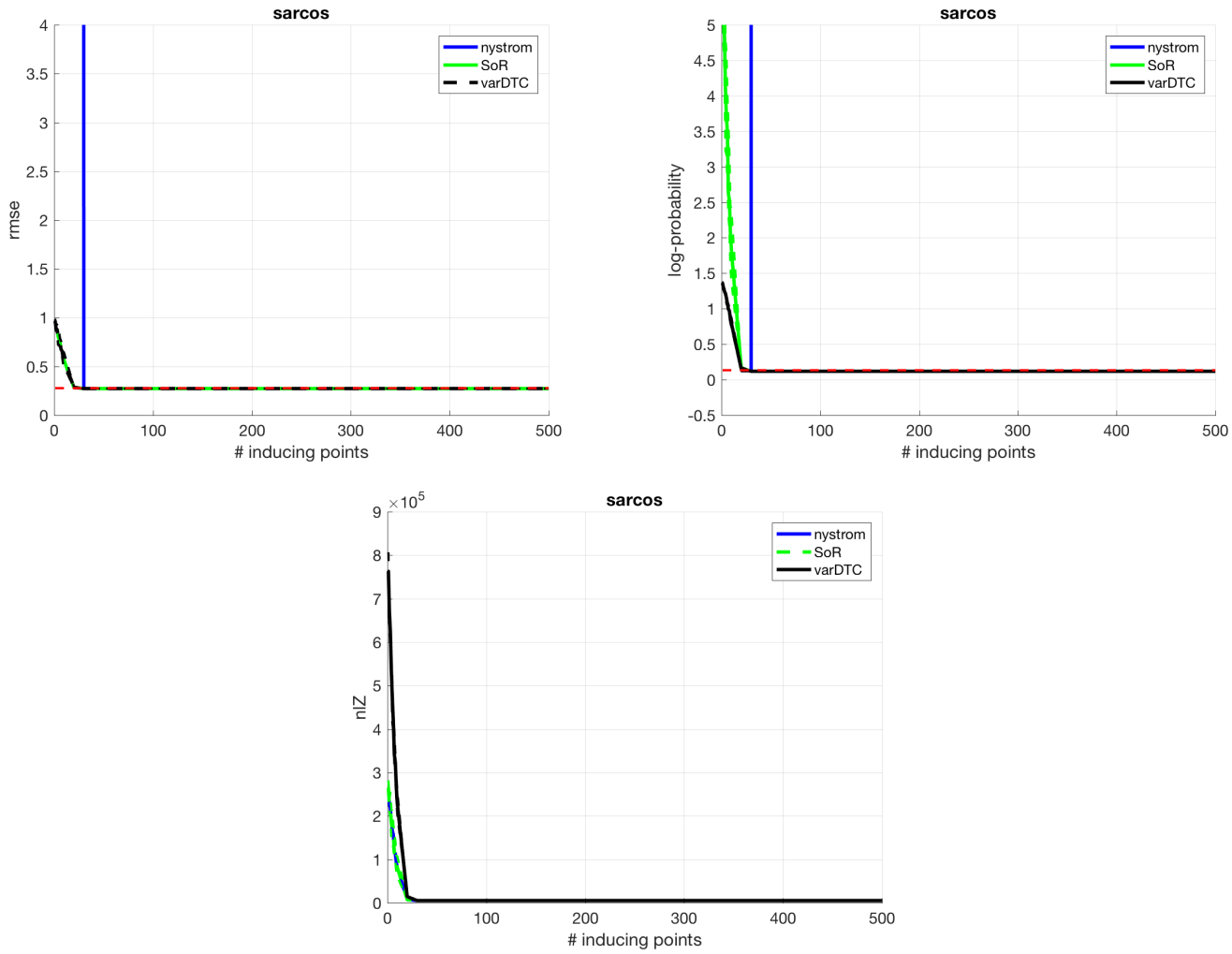


Figure 18: This figure shows the performance of the three low-rank approximation kernel algorithms as a function of the number of inducing points on **Sarcos** dataset. In this experiment the kernel used is the **LINiso** kernel. Each figure shows a different loss function. The red dotted line shows the performance of exact GP regression. Dashed line represent the 25th and 75th percentile while solid lines shows the median. Linear kernels capture only very simple characteristics of the data, hence the are not difficult to approximate and leads to poorer performance than Square Exponential kernels.

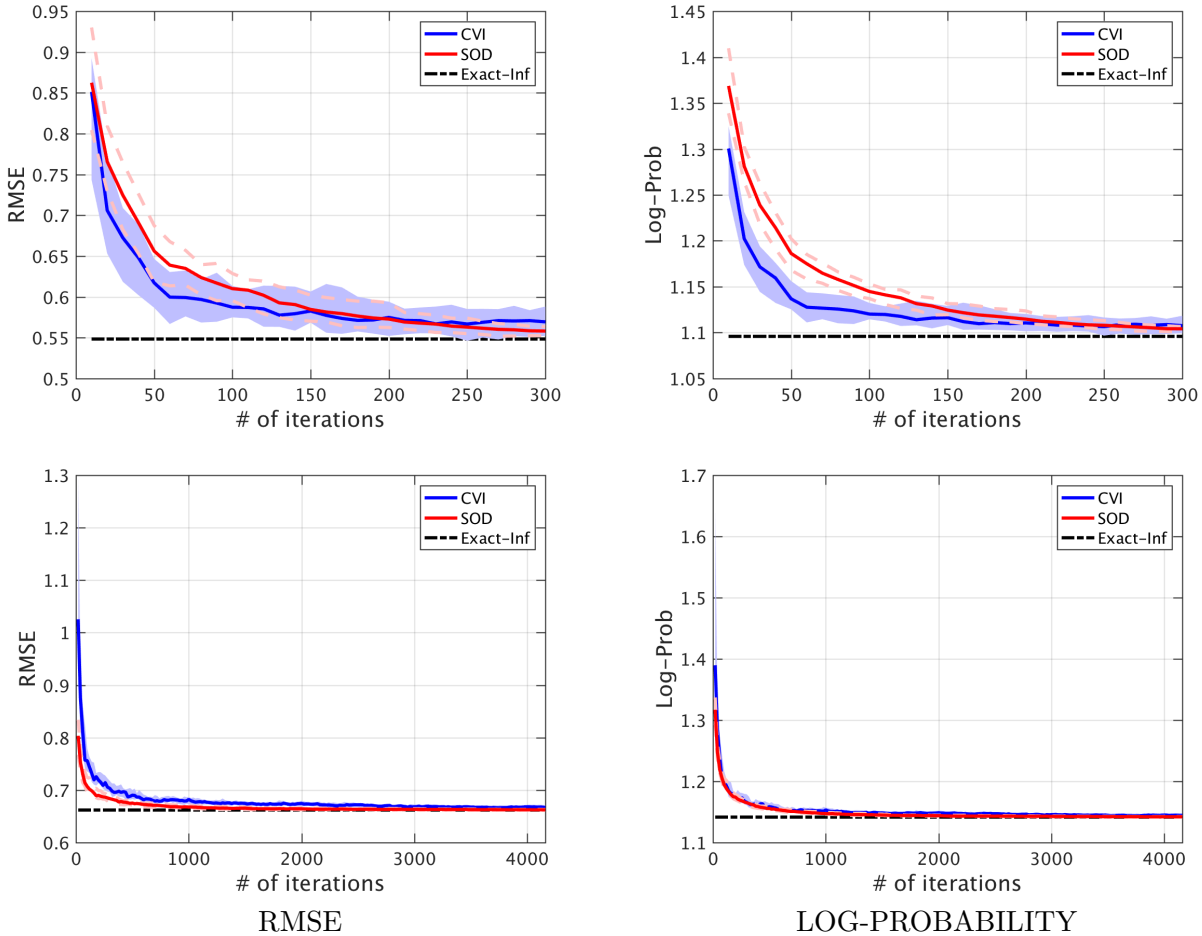


Figure 19: The first raw of plots shows the results obtained on **Yacht Hydrodynamics** dataset (**154** training points), the second raw shows the results obtained on **Abalone** dataset (**3132** training points). The blue shaded area covers the 25th-75th percentiles of the CVI performance. The red dashed lines represent the 25th and 75th percentile of SoD performance. The black dashed line shows the performance of exact GPR.

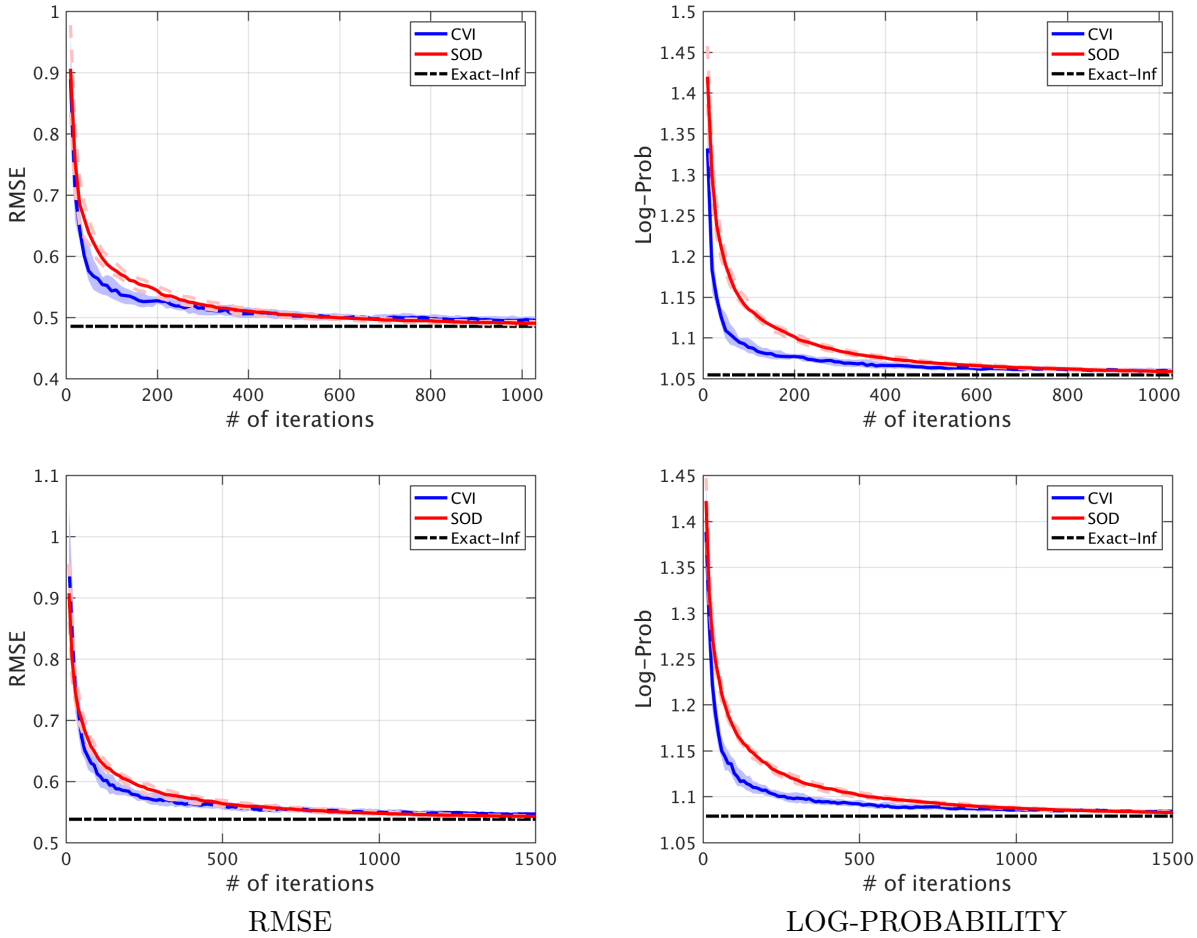
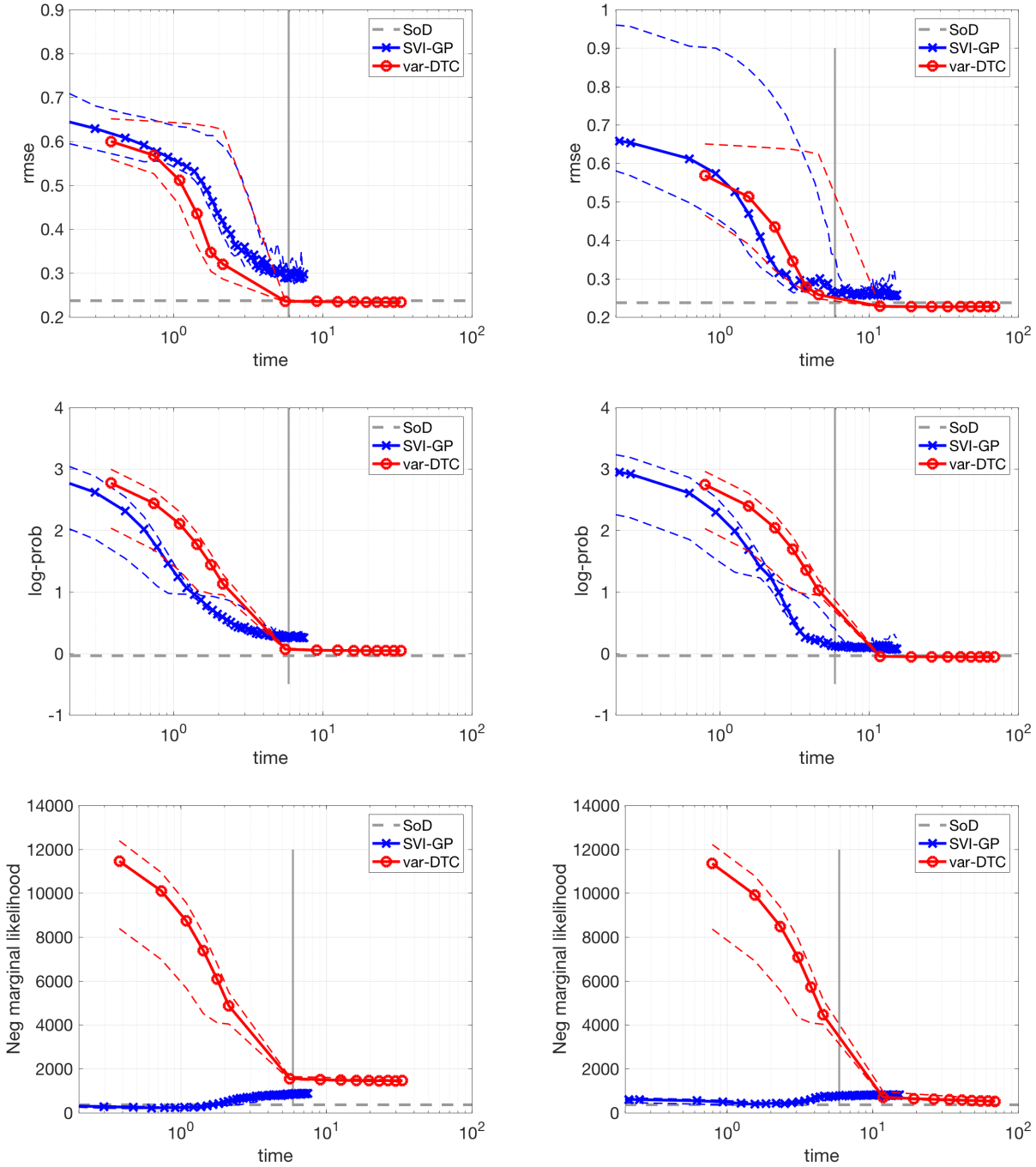


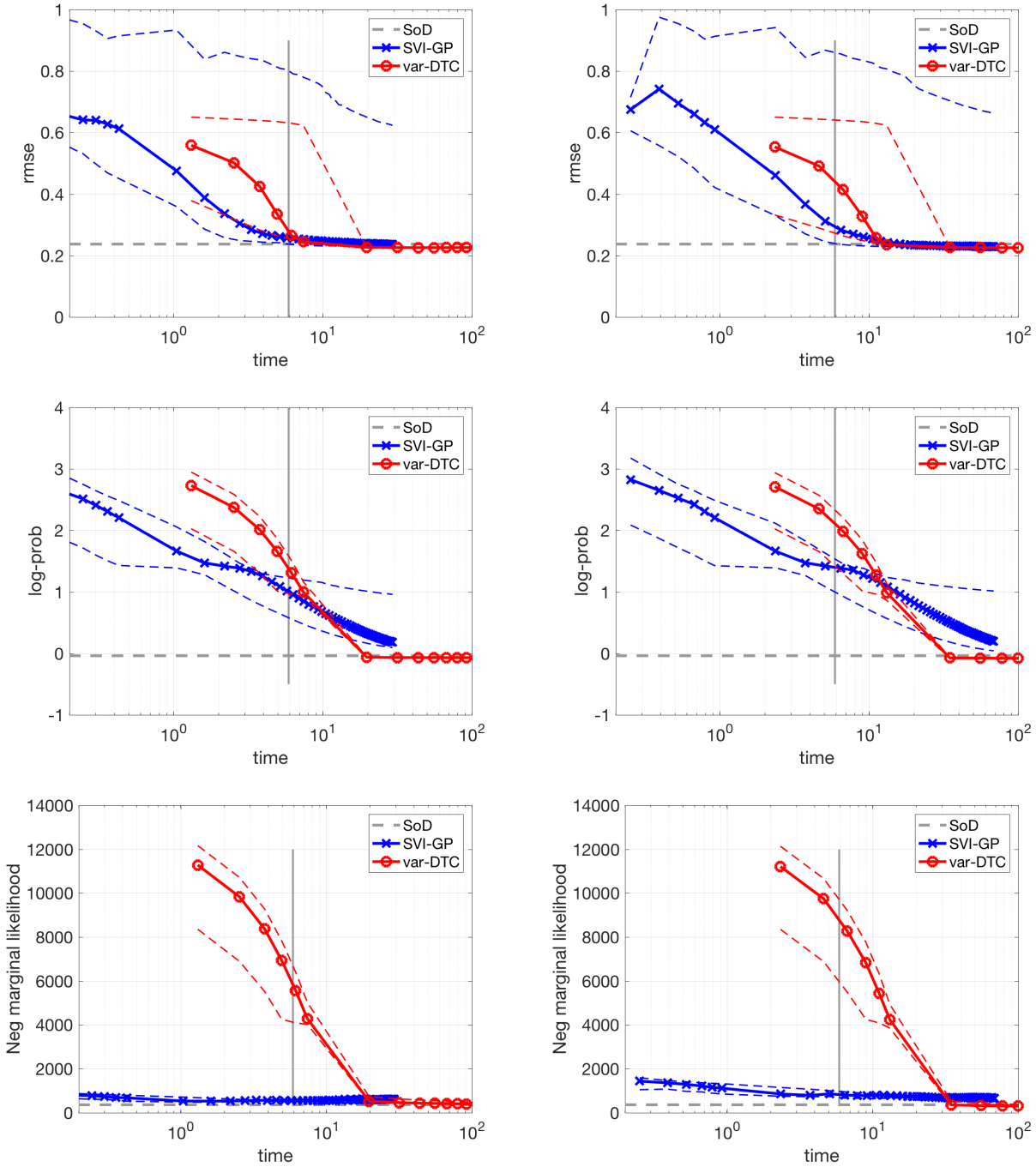
Figure 20: The first raw of plots shows the results obtained on **Concrete Compressive Strength** dataset (**515** training points), the second raw shows the results obtained on **Airfoil** dataset (**752** training points). The blue shaded area covers the 25th-75th percentiles of the CVI performance. The red dashed lines represent the 25th and 75th percentile of SoD performance. The black dashed line shows the performance of exact GPR.



100 inducing points

200 inducing points

Figure 21: This figure shows results obtained on **Pumadyn-8nm** dataset, each row of figures shows a different loss function. The left column shows results obtained using 100 inducing points and the right columns shows results obtained using 200 inducing points. The vertical grey line shows the time required to compute the SoD predictions using quasi-Newton optimization algorithm from *MinFunc* library. We see that varDTC needs much less steps than SVI-GP to converge. We also note improvement for both methods when increasing the number of inducing points.



300 inducing points

500 inducing points

Figure 22: This figure shows results obtained on **Pumadyn-8nm** dataset, each row of figures shows a different loss function. The left column shows results obtained using 100 inducing points and the right columns shows results obtained using 200 inducing points. The vertical grey line shows the time required to compute the SoD predictions using quasi-Newton optimization algorithm from *MinFunc* library. We see that varDTC needs much less steps than SVI-GP to converge. We also note improvement for both methods when increasing the number of inducing points.

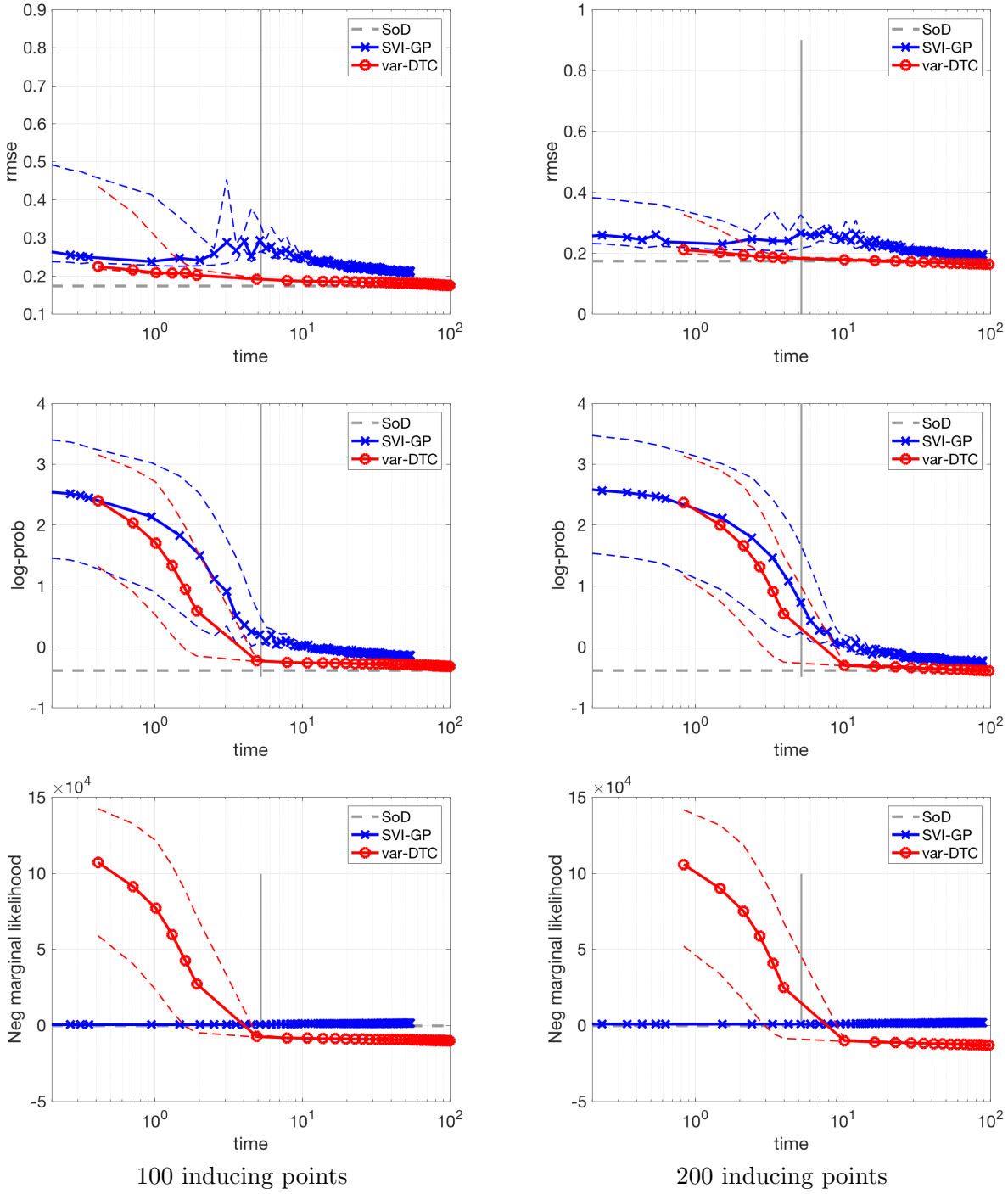
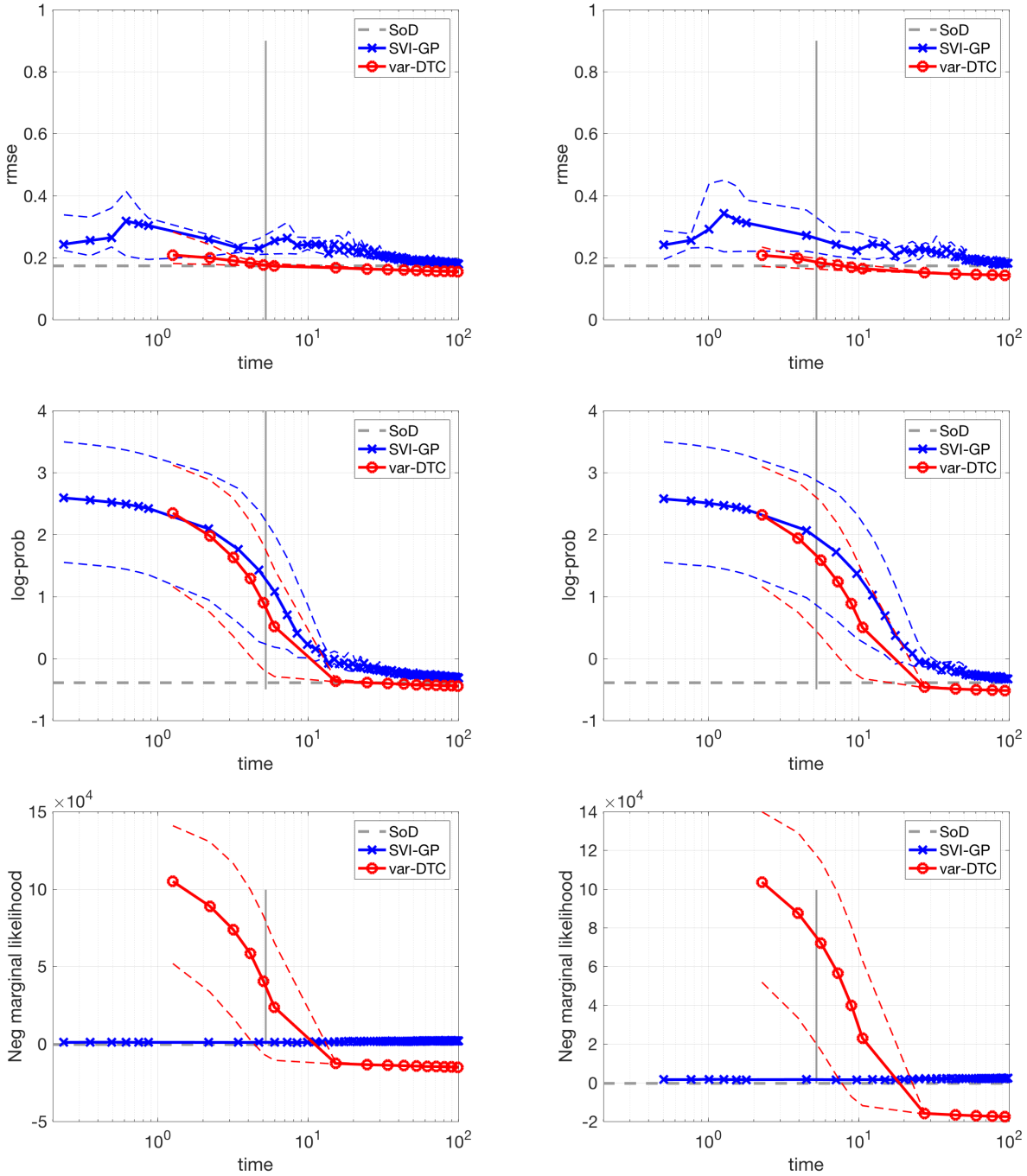


Figure 23: This figure shows results obtained on **Sarcos** dataset, each row of figures shows a different loss function. The left column shows results obtained using 100 inducing points and the right columns shows results obtained using 200 inducing points. The vertical grey line shows the time required to compute the SoD predictions using quasi-Newton optimization algorithm from *MinFunc* library. We see that varDTC needs much less steps than SVI-GP to converge. We also note improvement for both methods when increasing the number of inducing points.



300 inducing points

500 inducing points

Figure 24: This figure shows results obtained on **Sarcos** dataset, each row of figures shows a different loss function. The left column shows results obtained using 100 inducing points and the right columns shows results obtained using 200 inducing points. The vertical grey line shows the time required to compute the SoD predictions using quasi-Newton optimization algorithm from *MinFunc* library. We see that varDTC needs much less steps than SVI-GP to converge. We also note improvement for both methods when increasing the number of inducing points.