BPINN-Eikonal-Inverse-UQ

Uncertainty quantification of conduction velocities from noisy activation maps using Bayesian Physics-Informed Neural Networks

Daniele Ceccarelli
Supervisor: Prof. Manzoni
Co-supervisor: Dr. Pagani

ADVANCED PROGRAMMING FOR SCIENTIFIC COMPUTING
Mathematical Engineering, Politecnico di Milano

A.A. 2019/2020

Inverse UQ problem

Inverse UQ problem

Starting from some scattered and noisy measurements $\boldsymbol{D}$ reconstruct the posterior Probability Density Function of unknow parameters $\boldsymbol{\theta}$ of a given partial differential equation.

Bayes Rule

$$\mathcal{P}(\boldsymbol{\theta}|\boldsymbol{D}) = \frac{P(\boldsymbol{D}|\boldsymbol{\theta})\mathcal{P}(\boldsymbol{\theta})}{\mathcal{P}(\boldsymbol{D})}$$

New solution

Limitations: Classical methods involving PDE models require to solve multiple times ($O(10^5/10^6)$) the PDE to sample the posterior distribution.

Our solution: **Bayesian PINN** $\rightarrow$ a method to directly approximate the posterior distribution of a physics problem with Neural Networks.

Physics-Informed Neural Networks (PINN)

PINN is a deterministic approach to solve forward or parameter estimation problems.

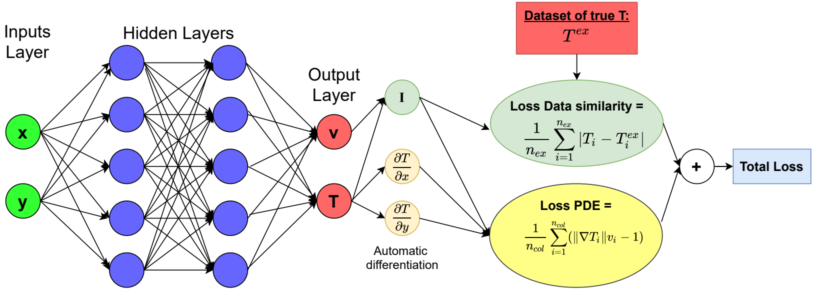


Figure 1: Example of PINN Architecture

Bayesian PINN

Extension of PINN to a Bayesian framework in order to reconstruct the posterior distribution of our parameters.

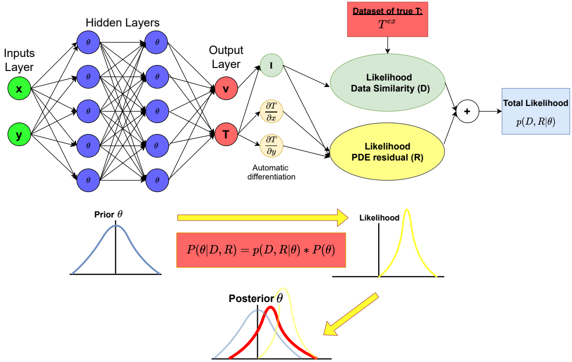


Figure 2: BPINN Architecture

Datasets

To evaluate the Bayes formula in a NN we need 2 datasets:

- ***D***: Dataset of measurements of T, dim=n_{ex}, affected by Gaussian noise:

$$\hat{T}_i = T_i + \epsilon_i, \quad \epsilon_i \sim \mathcal{N}(0, \sigma^2) \;\; \forall i = 1, \ldots, n_{ex}.$$

- ***R***: Dataset of PDE residuals, dim=n_{coll}, collocations points where we enforce the validity of PDE.

Bayesian model: Likelihood functions

Let $\boldsymbol{\theta}$ be the vector that contains all the weights in the NN.

Likelihood function:

$$\mathcal{P}(\boldsymbol{D}^{\boldsymbol{\theta}}|\boldsymbol{\theta}, \sigma_D) \sim \mathcal{N}(\boldsymbol{D}, \Sigma_D)$$

where $\Sigma_D = \sigma_D^2 \mathbb{I}$, and σ_D can be a fixed value or an hyperparameter.

$$\mathcal{P}(\boldsymbol{R}^{\boldsymbol{\theta}}|\boldsymbol{\theta}, \sigma_R) \sim \mathcal{N}(\boldsymbol{R}, \Sigma_R)$$

where $\Sigma_R = \sigma_R^2 \mathbb{I}$, and also in this case σ_R can be fixed or an hyperparameter.

Bayesian model: Priors

We have to specify a **prior** distribution on parameters vector $\boldsymbol{\theta}$, for instance a t-student:

$$\mathcal{P}(\theta_j) \sim StudentT(\mu^*, \lambda^*, \nu^*) \quad \forall j = 1, \ldots, dim^{\boldsymbol{\theta}}$$

If we consider both σ_D and σ_R trainable, we need a prior distribution on them, for instance an Inverse Gamma:

$$\mathcal{P}(\sigma_D^2) \sim Inv - Gamma(\alpha_1, \beta_1)$$

$$\mathcal{P}(\sigma_R^2) \sim Inv - Gamma(\alpha_2, \beta_2).$$

Bayesian Methods

The **posterior** distribution can be computed as:

$$
\begin{aligned}
\mathcal{P}(\boldsymbol{\theta}, \sigma_D, \sigma_R | \boldsymbol{D}^\theta, \boldsymbol{R}^\theta) \;\; \alpha \;\; & \mathcal{P}\left(\boldsymbol{D}^\theta | \boldsymbol{\theta}, \sigma_D\right) \mathcal{P}\left(\boldsymbol{R}^\theta | \boldsymbol{\theta}, \sigma_R\right) \mathcal{P}\left(\boldsymbol{\theta}\right) \mathcal{P}\left(\sigma_D\right) \mathcal{P}\left(\sigma_R\right) \\
\alpha \;\; & \mathcal{N}\left(\boldsymbol{D}^\theta; \boldsymbol{D}, \Sigma_D\right) \mathcal{N}\left(\boldsymbol{R}^\theta; 0, \Sigma_R\right) StudentT(\boldsymbol{\theta}; \mu^*, \lambda^*, \nu^*) \\
& IG\left(\sigma_D^2; \alpha_1, \beta_1\right) IG\left(\sigma_R^2; \alpha_2, \beta_2\right)
\end{aligned}
$$

To sample from this posterior we use two different methods:

- **HMC**: Hamiltonian Monte Carlo, a classical MCMC method,
- **SVGD**: Stein Variation Gradient Descent.

HMC

Goal: update weights to approximate the posterior distribution.

- Initialize $\boldsymbol{\theta}^{t_0}$, fix N, M, L and dt;
- for every k in $1, \ldots, N$:
  1. Sample $\mathrm{r}^{t_{k-1}} \sim \mathcal{N}(0, \mathbb{I})$
  2. $(\boldsymbol{\theta}_0, \mathrm{r}_0) = (\boldsymbol{\theta}^{t_{k-1}}, \mathrm{r}^{t_{k-1}})$
  3. for i in $0, \ldots, (L-1)$:

$$\begin{aligned} \mathrm{r}_i &= \mathrm{r}_i - \frac{dt}{2} \nabla U(\boldsymbol{\theta}_i) \\ \boldsymbol{\theta}_{i+1} &= \boldsymbol{\theta}_{\boldsymbol{i}} + dt \mathrm{r}_i \\ \mathrm{r}_{i+1} &= \mathrm{r}_i - \frac{dt}{2} \nabla U(\boldsymbol{\theta}_{i+1}) \end{aligned}$$

  4. sample $p \sim Uniform(0, 1)$
  5. compute $\alpha = \min(1,\ \exp(H(\boldsymbol{\theta}_L, \mathrm{r}_L) - H(\boldsymbol{\theta}^{t_{k-1}}, \mathrm{r}^{t_{k-1}})))$
  6. If $p \geq \alpha$, then $\boldsymbol{\theta}^{t_k} = \boldsymbol{\theta}_{\boldsymbol{L}}$, else $\boldsymbol{\theta}^{t_k} = \boldsymbol{\theta}^{t_{k-1}}$

and finally compute all the statistics we need using $\{\boldsymbol{\theta}^{t_i}\}_{i=N-M+1}^{N}$.

Application

Inverse UQ problem

Starting from some scattered and noisy measurements $\{\hat{T}_i\}_{i=1}^{n_{ex}}$ reconstruct the posterior Probability Density Function of conductivity tensor $\boldsymbol{M}(\boldsymbol{x})$ using equation:

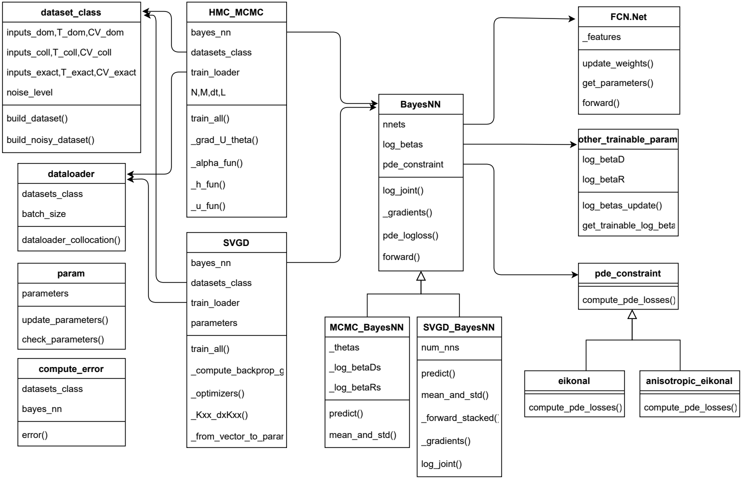
$$\begin{cases} \sqrt{\nabla T(\boldsymbol{x})^T \boldsymbol{M}(\boldsymbol{x}) \nabla T(\boldsymbol{x})} = 1 & in \;\; \Omega, \\ T(\hat{\boldsymbol{x}}_i) = 0 & \forall i = 1, \dots, N_S. \end{cases} \tag{1}$$

If $\boldsymbol{M}(\boldsymbol{x}) = v(\boldsymbol{x})^2 \mathbb{I}_d$, for a suitable conduction velocity $v : \mathbb{R}^d \to \mathbb{R}$, equation (1) becomes:

$$\begin{cases} \|\nabla T(\boldsymbol{x})\| = \dfrac{1}{v(\boldsymbol{x})} & in \;\; \Omega, \\ T(\hat{\boldsymbol{x}}_i) = 0 & \forall i = 1, \dots, N_S. \end{cases} \tag{2}$$

Code organization

Goal: flexible implementation of Bayesian PINN

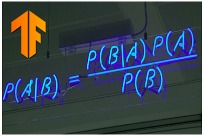


External libraries

This implementation is based on **Python** (3.8.5)

External libraries:

- Tensorflow 2
- Tensorflow-probability
- Numpy
- Matplotlib



Fully Connected Networks

Attribute:
self._features : a neural network implemented adding tf.keras.layers.Dense layers in a tf.keras.Sequential() model. All the layers have a *Swish* activation function and (glorot_uniform,zeros) initializer for (weights,bias) parameters.
Methods:

- **forward(self, x)**: return the forward pass of input x in the network (self._features(x));
- **get_parameters(self)**: return all the trainable parameters in the NN in a list;
- **update_weights(self, param)**: update the weights in all the layers.

Bayesian PINN

Attributes:

- **self.nnets** : neural network (or a list of nns for SVGD) to be trained (parameters vector $\boldsymbol{\theta}$);
- **self.log_betas** : $\log \frac{1}{\sigma_D^2}$ and $\log \frac{1}{\sigma_R^2}$;
- **self.pde_constraint** : compute the specific PDE residual.

Methods:

- **log_joint(self, output, target)**: compute the log likelihood of data similarity and log prior of $\boldsymbol{\theta}$ (and σ_D if trainable);
- **_gradients(self, inputs)**: compute the gradients wrt inputs (using Automatic Differentiation);
- **pde_logloss(self, inputs)**: compute the log loss of PDE constraint (and log prior of σ_R if trainable);
- **predict(self, inputs)** and **mean_and_std(self, inputs)**: prediction, mean and std on inputs.

HMC implementation

Attributes:

- **self.bayes_nn**: the BPINN object we want to train with HMC;
- **self.dataset_class** and **self.train_loader** : datasets and data loader classes;
- **parameters** of HMC.

Methods:

- **train_all(self)**: method called from main to start the training;
- **_u_fun(self,...)**: compute $U(\theta) = -(\log(P(\boldsymbol{D}|\theta)) + \log(P(\boldsymbol{R}|\theta)) + \log(P(\theta)));$
- **_Grad_U_theta(self,...)**: compute gradient of $U(\theta)$ wrt θ;
- **_alpha_fun(self,...)**: compute α for accept/reject step.

1D Isotropic

1D Isotropic example:

$$\begin{cases} \left|\dfrac{\partial T(x)}{\partial x}\right| = \dfrac{1}{v(x)}, \quad in\ \Omega = (0,1), \\ T(0) = 0. \end{cases} \tag{3}$$

with the following solutions:

$$\begin{aligned} T(x) =& 1 - e^{-2x} \\ v(x) =& \frac{1}{2}e^{2x}. \end{aligned} \tag{4}$$

Inverse UQ varying the number of exact data (10,20 and 40) and noise level on it (0.01,0.05 and 0.10)

1D Isotropic

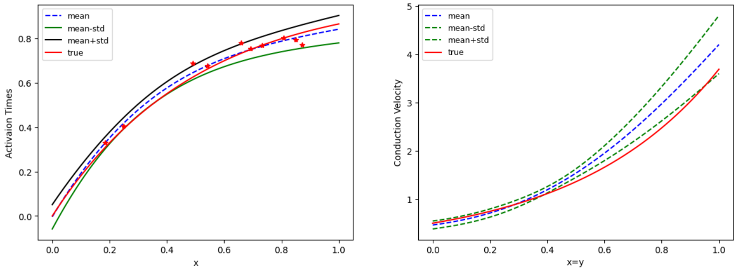


Figure 3: Activation times and conduction velocity in 1D Isotropic example with 10 data, noise level 0.05

1D Isotropic

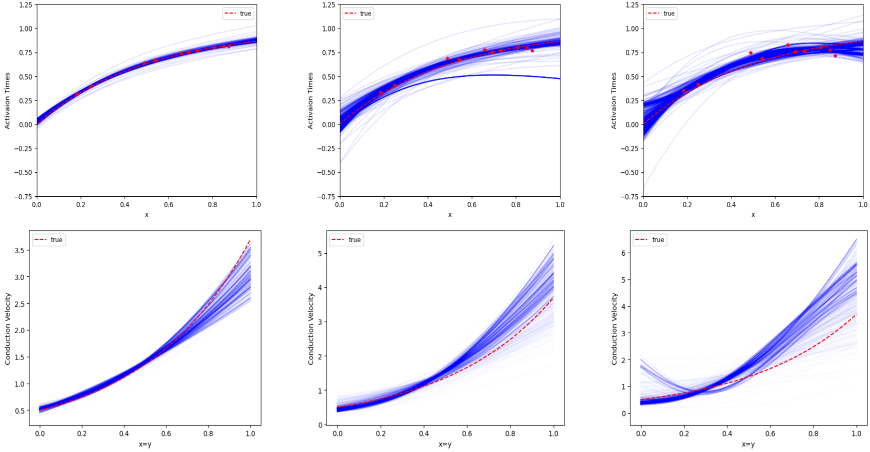


Figure 4: Comparison with 10 data and 0.01/0.05/0.10 noise level, samples from posterior distribution

2D Anisotropic

$$M(x) = \begin{bmatrix} a(x,y) & -c(x,y) \\ -c(x,y) & b(x,y) \end{bmatrix}, \quad \begin{cases} b(x,y) = 2a(x,y) \\ c(x,y) = a(x,y) \end{cases} \tag{5}$$

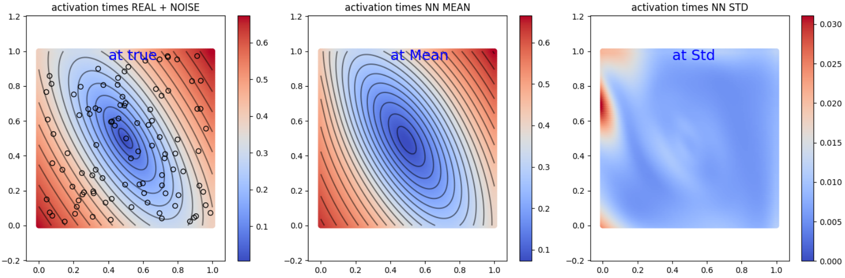


Figure 5: T(x,y) (true, mean and std of posterior distribution) in 2D Anisotropic example

2D Anisotropic

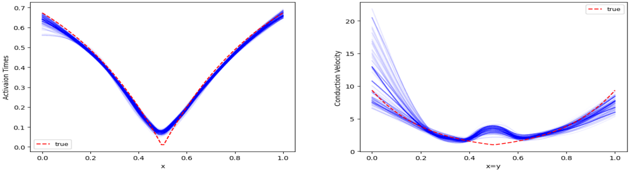


Figure 6: T(x,y) and a(x,y) on the line y=x

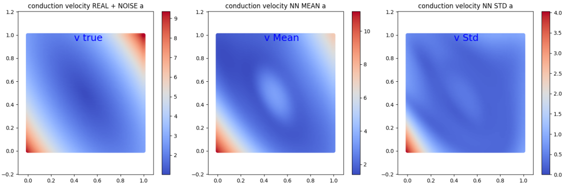


Figure 7: a(x,y) (true, mean and std) in 2D Anisotropic example

Prolate 3D

Inverse UQ problem

Starting from some measurements of T $\{\hat{T}_i\}_{i=1}^{n_{ex}}$, compute the posterior PDF of $\boldsymbol{\theta}$:

$$\mathcal{P}(\boldsymbol{\theta}|\boldsymbol{D}, \boldsymbol{R})$$

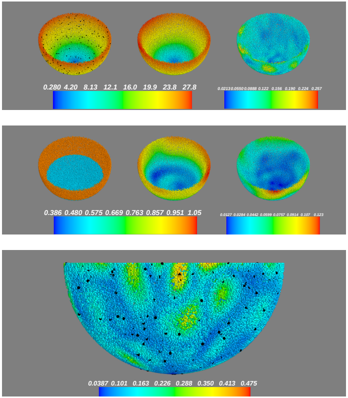


Figure 8: Activation times and conduction velocity (true, mean and std of post. distr.) in 3D prolate example

Graph Execution

Tensorflow 2 has the possibility of use both Graph or Eager execution.

- **Eager execution**: Natural control flow and easy to debug;
- **Graph execution**: Instantiate a Graph for computation, difficult to debug but faster.

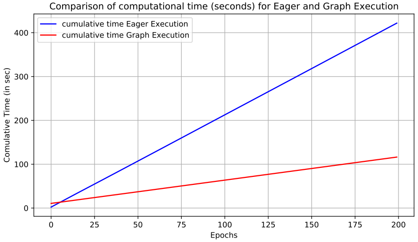
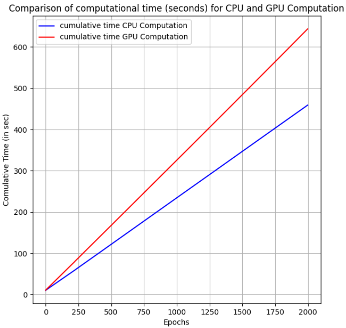


Figure 9: Computational time comparison for Eager and Graph execution

GPU accelerator



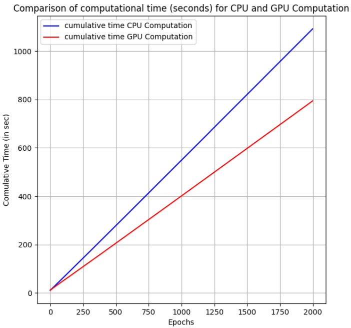


Figure 10: Computational time comparison for CPU and GPU in a small network (3/20 architecture) and a big network (5/100 architecture)

Conclusions

- BPINN gives an efficient and accurate inverse problem solution of Eikonal equation;
- Inverse UQ problems: preliminary results are capturing variance trend but we need to investigate more (the relation with prior distributions and likelihood functions);
- We test the model in different numerical experiments (up to 3D Isotropic and 2D Anisotropic);
- Flexible implementation, could be used with minor changes also to solve Inverse UQ for other PDEs.

Further developments

- Explore the relationship with **different Bayesian models** (priors and likelihoods);
- Implement different MCMC methods (for instance **NUTS**);
- **Active learning** algorithm to dynamically minimize the uncertainty;
- **Transfer learning** with subdomains;
- Anisotropy and **more realistic** scenarios.

References

- M. Raissi and P. Perdikaris and G.E. Karniadakis, “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations”.
- Liu Yang and Xuhui Meng and George Em Karniadakis, “B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data”.
- Sahli Costabal, Francisco and Yang, Yibo and Perdikaris, Paris and Hurtado, Daniel E. and Kuhl, Ellen, “Physics-informed neural networks for cardiac activation mapping”.
- Sun, Luning and Wang, Jian-Xun, ”Physics-constrained Bayesian neural network for fluid flow reconstruction with sparse and noisy data”

Appendix: SVGD

- Initialize N neural networks weights $\boldsymbol{\theta^i}$ $\forall i = 1, \ldots, N$;
- For every epochs $e = 1, \ldots, E$:
  1. Compute the log posterior $L(\boldsymbol{\theta}^i)$ $\forall i = 1 \ldots, N$
  2. Compute the gradients $\nabla_{\boldsymbol{\theta}^i} L(\boldsymbol{\theta}^i)$ by back-propagation
  3. Compute

$$\phi(\boldsymbol{\theta}^i) = \frac{1}{N} \sum_{j=1}^{N} \left[k(\boldsymbol{\theta}^i, \boldsymbol{\theta}^j) \nabla_{\boldsymbol{\theta}^i} L(\boldsymbol{\theta}^i) + \nabla_{\boldsymbol{\theta}^i} k(\boldsymbol{\theta}^i, \boldsymbol{\theta}^j) \right] \quad \forall i = 1 \ldots, N \tag{6}$$

  4. Update $\boldsymbol{\theta}^i = \boldsymbol{\theta}^i + \epsilon \phi(\boldsymbol{\theta}^i)$ or use a Stochastic Gradient Descent method, $\forall i = 1, \ldots, N$.

Appendix: Posterior PDF

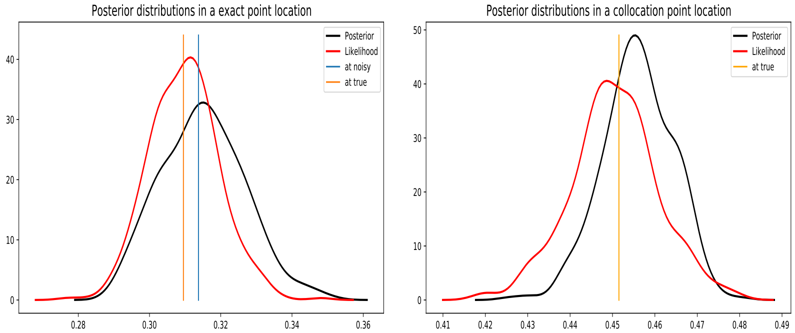
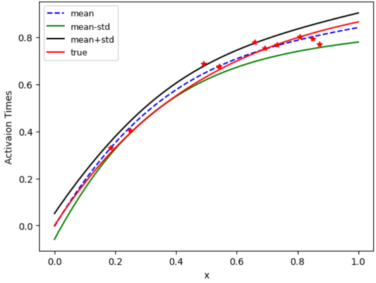


Figure 11: Posterior PDF and Likelihood on Activation times locations

Appendix: Influence of Eikonal



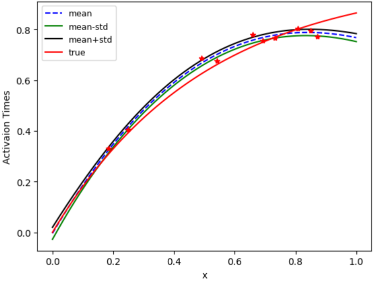


Figure 12: Comparison of activation times with 10 exact data, noise_lv = 0.05, "With Eikonal" on the left, "Without Eikonal" on the right

Appendix: Posterior of σ_D

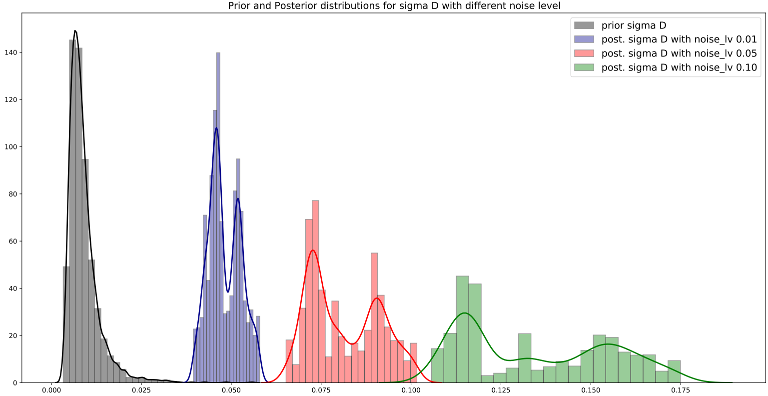


Figure 13: σ_D prior and posteriors for noise level $= 0.01, 0.05, 0.10$