Imperial College London

Department of Earth Science and Engineering

MSc Applied Computational Science and Engineering

Independent Research Project - Project Plan

Bayesian Machine Learning for Quantifying Uncertainty in Surrogate Modelling

by

Archie Luxton

In collaboration with Arup
8 Fitzroy Street, London, United Kingdom, W1T 4BJ

Email: archie.luxton21@imperial.ac.uk
GitHub username: asce-aol21
Repository: https://github.com/ese-msc-2021/irp-aol21

Supervisors:
Dr. Ramaseshan Kannan (Arup)
Dr. Gerard Gorman (Imperial College London)

1st July 2022
(Extension to 4th July 2022)

Abstract

Surrogate modelling is a fast-growing application area in scientific machine learning. The goal of machine learning-based surrogate modelling is to learn from high fidelity simulation data to train a model on which it's computationally cheaper to make inference. This can be useful in early stage exploration of the design space where high fidelity results aren't really necessary. The challenge however with ML based surrogate modelling is that we don't know how accurate our predictions are. We can, however, quantify uncertainties in the predictions using Bayesian inference, which will enable us to accept or reject predictions in the surrogate with confidence. This project will develop and evaluate a Bayesian statistical framework applied to a neural network-based surrogate model. The intended purpose of the surrogate model is to predict seismic behaviour on buildings based on results from vibrational analysis of finite element models of buildings.

1 Introduction and Literature Review

1.1 Bayesian Inference and MCMC

Bayesian inference is concerned with updating belief after considering new evidence. A Bayesian approach interprets a probability as an individual's degree of belief of an event occurring, as opposed to the Frequentist approach which interprets a probability as an objective long-run frequency of an event occurring [1] - i.e. the limit of the frequency of an event after many trials.

At the core of Bayesian inference is Bayes' Theorem. Given some model parameters θ, independent variables $\mathbf{X}$ and dependent variables $\mathbf{y}$, Bayes' theorem is given as:

$$P(\theta|y, \mathbf{X}) = \frac{P(y|\theta)P(\theta)}{P(\mathbf{X})}$$

Where $P(\theta|y, \mathbf{X})$ is the **posterior**, $P(y|\theta)$ is the **likelihood**, $P(\theta)$ is the **prior**, and $P(\mathbf{X})$ is the **evidence** (sometimes also called the **normalisation** term [7]). If we are not able to form a **conjugate prior** [13] [9], the evidence term is intractable - i.e. we cannot form a closed-form analytic solution [5]. This project will be focusing on the scenario where an analytical solution cannot be formed, so a posterior sampling method should be used - such as the **Metropolis Hastings Monte-Carlo Markov Chain** (MCMC), introduced in 1970 [4].

1.2 Surrogate Models

Finite Element Analysis (FEA) software suites such as *Oasys GSA* that are used for engineering structural analysis can be extremely computationally complex, so there is an appetite for *surrogate models* that can provide similar results but with reduced run times.

Deep learning based surrogate models for structural and stress analysis are documented in literature [12] [6], but the number of highly cited works on Bayesian deep learning-based surrogate models is very limited, with much of the work being produced by the Center for Informatics and Computational Science at the University of Notre Dame [14] [15], who apply the techniques to solving PDEs.

2 Problem Description and Objectives

The focus of this project will be developing a Bayesian statistical framework that can be applied to a range of surrogate models. The project has thus far been concerned with the application of Bayesian statistics to regression, and in time will progress towards more sophisticated models and applications.

A generalised Bayesian framework will be developed in Python that can be applied to surrogate models for engineering applications. There will be a focus on sustainability and usability to non-computer science specialists.

Though the final software will be primarily developed in Python, there may be optimisation carried out in C/C++ scripts. Given sufficient time, there may also be some work put into parallelising code to run on CPUs or even GPUs.

The resulting Bayesian framework will be applied to a surrogate model trained on time-series simulation data from *Oasys GSA*, investigating the effects of seismic forces on a large structure (pending confirmation of client confidentiality restrictions). The result will be a model that predicts the effect of the seismic forces on the structure, as well as accompanying uncertainties in that prediction. Hyperparameters will be tuned based on unseen validation data, and final testing will be carried out on unseen test data.

3 Progress to Date and Future Plan

To date, the focus of this project has been on Bayesian polynomial regression in 1D, finding the function $f(x)$ that maps a single independent variable ($x \in \mathbb{R}$) to a single dependent variable ($y \in \mathbb{R}$). Bayesian regression does not look for a point estimate of a target given a feature, but instead formulates the regression using probability distributions by determining the posterior distribution for the model parameters. For example considering a K-th degree polynomial function $f(x)$ with an additional noise term ϵ:

$$y = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 \dots \theta_K x^K + \epsilon$$

This can be generalised in matrix form as:

$$\mathbf{y} = \theta^{\mathbf{T}}\mathbf{X} + \epsilon$$

Or in expanded form:

$$\mathbf{y} = \begin{bmatrix} \theta_0 & \theta_1 & \cdots & \theta_K \end{bmatrix} \begin{bmatrix} \mathbf{x}^0 \\ \mathbf{x}^1 \\ \vdots \\ \mathbf{x}^K \end{bmatrix} + \epsilon$$

And for example a jth data point with a degree 3 polynomial:

$$y_j = \begin{bmatrix} \theta_0 & \theta_1 & \theta_2 & \theta_3 \end{bmatrix} \begin{bmatrix} 1 \\ x_j \\ x_j^2 \\ x_j^3 \end{bmatrix} + \epsilon$$

The term ϵ represents homoscedastic, aleatoric noise - in this example representing a measurement error. Therefore if we model ϵ as a normal distribution ($\epsilon \sim \mathcal{N}(0, \sigma^2)$), then we get the following *likelihood* term from Bayes' theorem:

$$p(y|\mathbf{X}, \theta) = \mathcal{N}(\mathbf{X}^{\mathbf{T}}\theta, \sigma^2)$$

Here, the Python module **BPRPy.py** (**B**ayesian **P**olynomial **R**egression **Py**thon) has been developed to perform Bayesian regression on synthetic or imported data sets. The class **BPR** contains methods to create a dummy data set(s) and perform the MCMC sampling (Metropolis-Hastings variant), and the *BPR_plots* class contains methods to produce all the plots seen below.

The x and y data sets are generated using the following relations:

- $\mathbf{x}$: $x \sim \mathcal{N}(0, \sqrt{5})$
- $\mathbf{y}$: $f(x) + \mathcal{N}(0, sd)$, where sd is user-defined

For the purposes of understanding, the MCMC sampling algorithm and plotting functions have all been implemented from scratch without using pre-built MCMC samplers. The plots have been designed to produce similar outputs to the PyMC3 [8] implementation, but with some differences in style to aid comprehension. It appears that this work represents the only Python implementation of Bayesian regression (using the MCMC algorithm) available online.

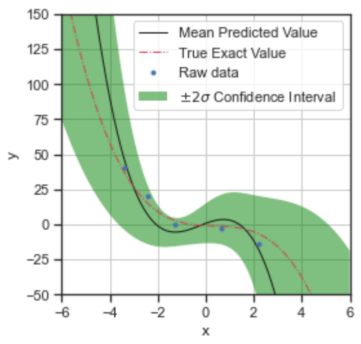
The MCMC method in the BPR class returns a *posterior chain*, which is the value of each parameter saved at every iteration. The chain is of size $(M, K+1)$, where M is the number of MCMC iterations and K is the number of model parameters θ. After accounting for burn-in, these posterior chains represent the *posterior distributions*.

A posterior prediction for y is made at any x by combining the posterior distributions together. For example, for a generic cubic function at $x = 5$:

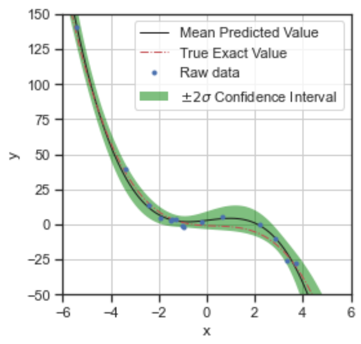
$$P(\theta|x=5) = P(\theta_0) + P(\theta_1|x=5) \times 5 + P(\theta_2|x=5) \times 5^2 + P(\theta_3|x=5) \times 5^3$$

This calculation is carried out across a range of x values, and then the mean value and the standard deviation of the resulting posterior distribution is used to produce the plots seen in Figure 1 and 2.

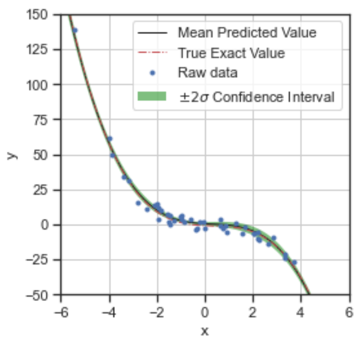
Figure 1 and 2 show the result of fixing the aleatoric uncertainty in the measured data (sd) and varying the number of data points N. As N increases, the uncertainty in the posterior prediction becomes smaller as would be expected. The region of highest confidence is where the density of training data is highest. Conversely, the confidence region expands in the out of domain (OoD) areas where there is no available training data.



(a) $N = 5$

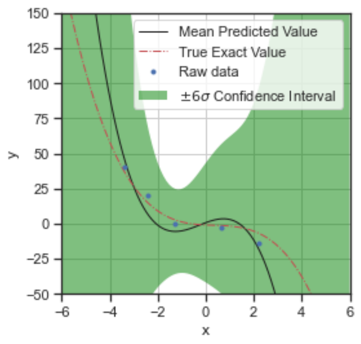


(b) $N = 15$

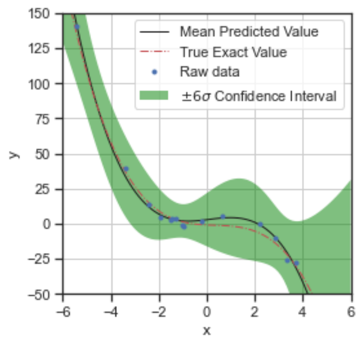


(c) $N = 50$

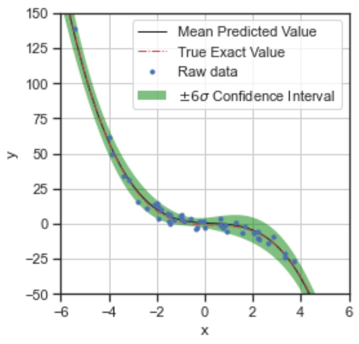
Figure 1: Cubic data set, $\pm 2\sigma$ confidence interval



(a) $N = 5$



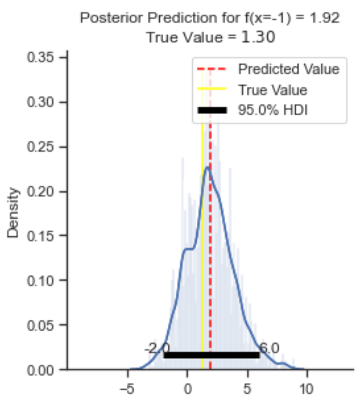
(b) $N = 15$



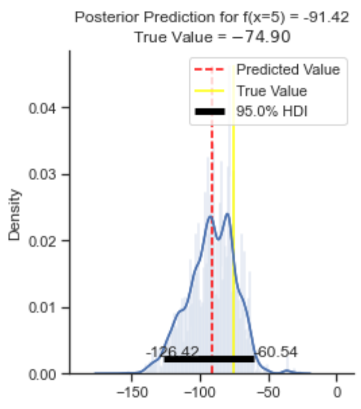
(c) $N = 50$

Figure 2: Cubic data set, $\pm 6\sigma$ confidence interval

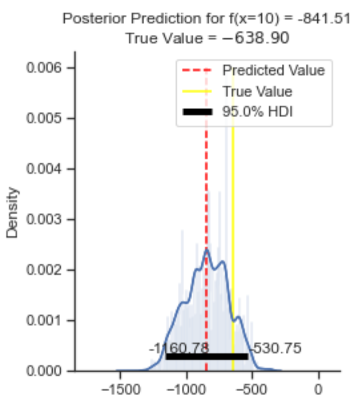
Figure 3a shows the posterior distribution of $f(x)$ for a point x within the range of observed data points and thus returns a prediction with a small variance: $f(x = -1)$ being between $(-2.0, 6.0)$ with a 95% certainty. Figures 3b and 3c show the posterior distributions for OoD predictions with $x = 5$ and $x = 10$ respectively. In the latter two plots the uncertainty is much higher, owing to the lack of training data in these regions.



(a) Prediction for $x = -1$
95% HDI range:
$(-2.0,\ 6.0)$



(b) Prediction for $x = 5$
95% HDI range:
$(-126.4,\ -60.5)$



(c) Prediction for $x = 10$
95% HDI range:
$(-1161.8,\ -530.8)$

Figure 3: Posterior predictions for $x = -1$, $x = 5$ and $x = 10$

If just the aleatoric noise (controlled by varying sd) is varied instead of N, the resulting certainty regions do not follow an expected pattern - this requires further investigation later in the project.

Figure 4 shows the belief plots - showing the posterior distribution - in the first column, the trace plots that show the value of each parameter in the chain after each iteration in the second column, and a plot showing the mean value of each parameter in every iteration in the third column. These plots demonstrate the purpose of 'burn-in': the MCMC algorithm takes some number of iterations to stabilise, so by discarding a chunk of iterations at the beginning, this unstable period can be ignored in the final posterior distribution.

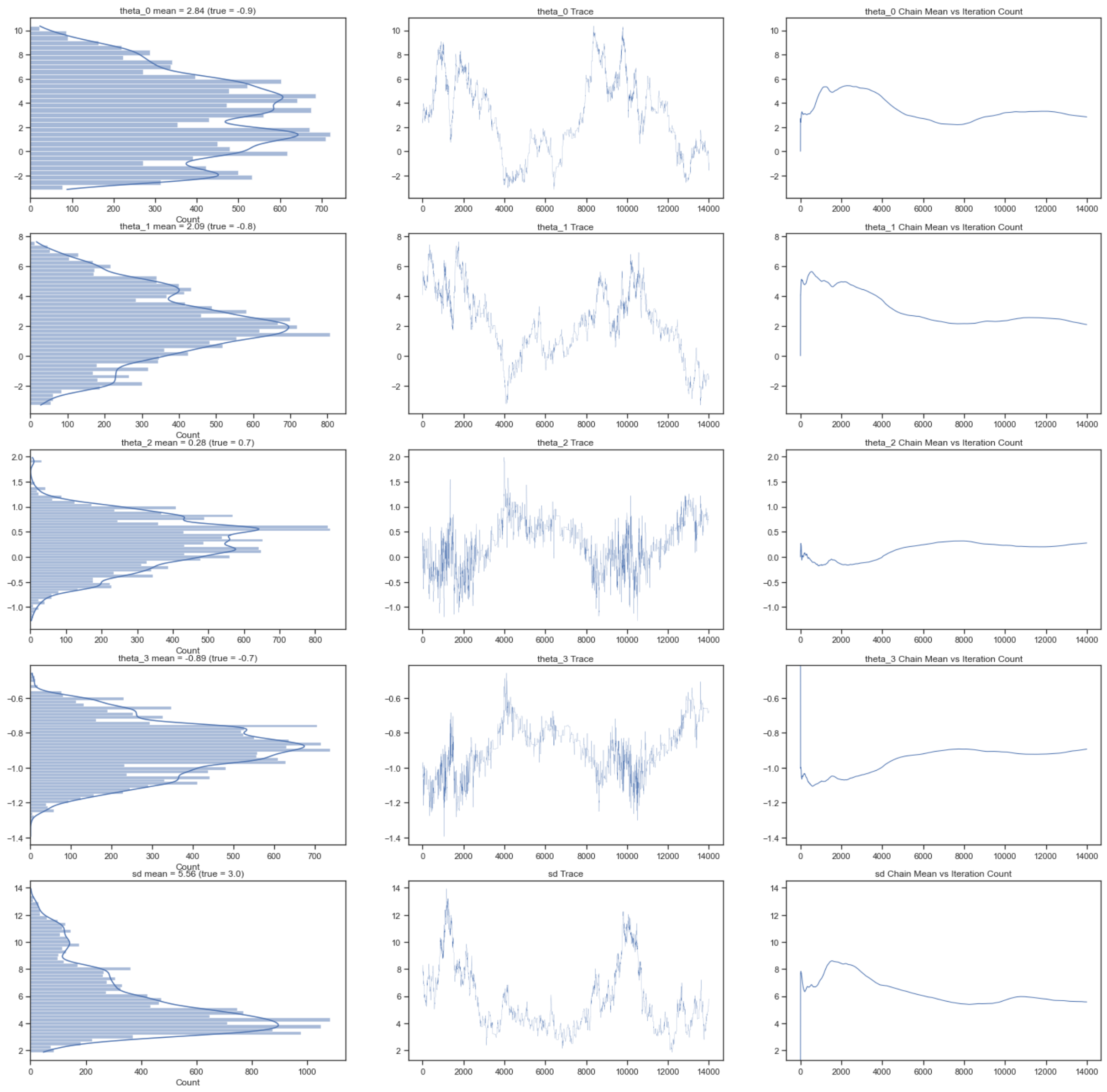


Figure 4: Belief, trace and mean value plots for $N = 15$, $sd = 10$

3.1 Future Plan

With the investigation into Bayesian 1D regression being concluded, the next phase of the project will briefly look at regression by other basis functions, for example hat/chapeau functions, Gaussians and/or splines. This approach may allow a better fit for different types of data [11].

After this, the developed Bayesian regression framework will be applied to a structural engineering application. A large number of simple structural models will be programmatically created in *Oasys GSA* and their natural frequency will be returned. Each model will have different input parameters that decides the geometry, such as:

- Number of bays (width)
- Number of bays (depth)
- Number of storeys (height)
- Bay width
- Storey height

A neural network based regression model will be developed that will map each of these parameters to a single output: *natural frequency*. The aim of the algorithm will be to predict the natural frequency of the structure based on *unseen* (OoD) data, and return an accompanying uncertainty in the prediction. This data may require pre-processing, e.g. using Principle Component Analysis (PCA) to reduce dimensionality, but this will be explored further at the time.

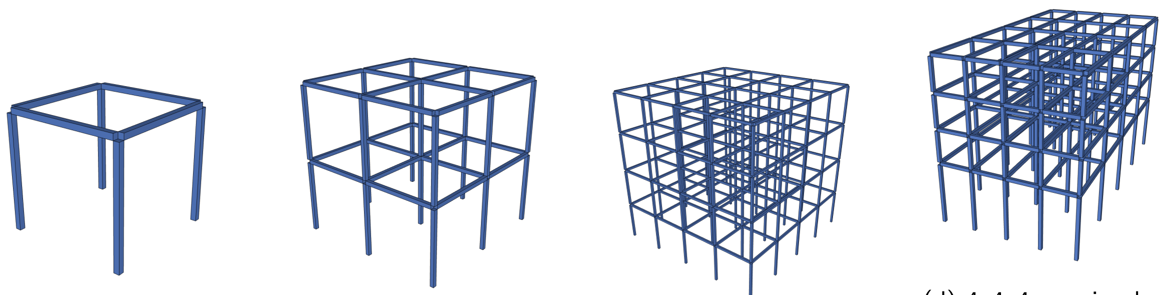

(a) 1x1x1 (b) 2x2x2 (c) 4x4x4 (d) 4x4x4, varying bay width

Figure 5: Examples of structure geometry to be explored

After this, Bayesian framework will be developed and applied to increasingly sophisticated ML-based surrogate models, starting at feed-forward networks and then following whatever methods show promise in modern literature.

In parallel, the computation time of the MCMC sampling should be reduced, either by using pre-built libraries such as PyMC3 [8], emcee [2] or Dynesty [10], or by developing an optimised version in C/C++ that gets called from Python.

Also in parallel, faster and more efficient methods than the Monte-Carlo Markov Chain (MCMC) sampling will be investigated, e.g. Integrated nested Laplace approximations [3], Conjugate-Prior scenarios [13], etc.

References

[1] Cameron Davidson-Pilon. *Bayesian methods for hackers: probabilistic programming and Bayesian inference*. Addison-Wesley Professional, 2015.

[2] Daniel Foreman-Mackey, Will M Farr, Manodeep Sinha, Anne M Archibald, David W Hogg, Jeremy S Sanders, Joe Zuntz, Peter KG Williams, Andrew RJ Nelson, Miguel de Val-Borro, et al. emcee v3: A python ensemble sampling toolkit for affine-invariant mcmc. *arXiv preprint arXiv:1911.07688*, 2019.

[3] Virgilio Gómez-Rubio. *Bayesian Inference with INLA*. Chapman and Hall/CRC Press. Boca Raton, FL., 2020.

[4] W Keith Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika, Vol. 57, No. 1 (Apr., 1970), pp. 97-109*, 1970.

[5] Will Koehrsen. Introduction to bayesian linear regression. `https://towardsdatascience.com/introduction-to-bayesian-linear-regression-e66e60791ea7`, 2018. Accessed on 06/05/2022.

[6] Liang Liang, Minliang Liu, Caitlin Martin, and Wei Sun. A deep learning approach to estimate stress distribution: a fast and accurate surrogate of finite-element analysis. *Journal of The Royal Society Interface*, 15(138):20170844, 2018.

[7] Michael Pyrcz University of Texas at Austin. Bayesian linear regression for subsurface data analytics in python. `https://github.com/GeostatsGuy/PythonNumericalDemos/blob/master/SubsurfaceDataAnalytics_BayesianRegression.ipynb`, 2021. Accessed on 10/06/2022.

[8] John Salvatier, Thomas V Wiecki, and Christopher Fonnesbeck. Probabilistic programming in python using pymc3. *PeerJ Computer Science*, 2:e55, 2016.

[9] Robert Schlaifer and Howard Raiffa. *Applied statistical decision theory*. Harvard University and MIT Press, 1961.

[10] Josh Speagle. Dynesty. `https://dynesty.readthedocs.io/`, 2017. Accessed on 15/06/2022.

[11] Jake VanderPlas. *Python Data Science Handbook: Essential Tools for Working with Data*. O'Reilly Media, Inc., 1st edition, 2016.

[12] Poojitha Vurtur Badarinath, Maria Chierichetti, and Fatemeh Davoudi Kakhki. A machine learning approach as a surrogate for a finite element analysis: Status of research and application to one dimensional systems. *Sensors*, 21(5):1654, 2021.

[13] Thomas Wiecki. Mcmc sampling for dummies. `https://twiecki.io/blog/2015/11/10/mcmc-sampling/`, 2015.

[14] Yinhao Zhu and Nicholas Zabaras. Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification. *Journal of Computational Physics*, 366:415–447, 2018.

[15] Yinhao Zhu, Nicholas Zabaras, Phaedon-Stelios Koutsourelakis, and Paris Perdikaris. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *Journal of Computational Physics*, 394:56–81, 2019.