




AutoEmulate: A Python package for semi-automated emulation

Martin A. Stoffel ^{1¶}, Bryan M. Li ^{1,2}, Kalle Westerling ¹, Sophie Arana ¹, Max Balmus ^{1,3}, Eric Daub ¹, and Steve Niederer ^{1,3}

¹ The Alan Turing Institute, London, United Kingdom ² University of Edinburgh, Edinburgh, United Kingdom ³ Imperial College London, London, United Kingdom ¶ Corresponding author

DOI: [10.21105/joss.07626](https://doi.org/10.21105/joss.07626)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Chris Vernon](#)  

Reviewers:

- [@willu47](#)
- [@lbi59](#)

Submitted: 03 December 2024

Published: 24 March 2025

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

Summary

Simulations are ubiquitous in research and application, but are often too slow and computationally expensive to deeply explore the underlying system. One solution is to create efficient emulators (also surrogate- or meta-models) to approximate simulations, but this requires substantial expertise. Here, we present AutoEmulate, a low-code, AutoML-style python package for emulation. AutoEmulate makes it easy to fit and compare emulators, abstracting away the need for extensive machine learning (ML) experimentation. The package includes a range of emulators, from Gaussian Processes, Support Vector Machines and Gradient Boosting Models to novel, experimental deep learning emulators such as Neural Processes ([Garnelo et al., 2018](#)). It also implements global sensitivity analysis as a common emulator application, which quantifies the relative contribution of different inputs to the output variance. Through community feedback and collaboration, we aim for AutoEmulate to evolve into an end-to-end tool for most emulation problems.

Statement of need

To understand complex real-world systems, researchers and engineers often construct computer simulations. These can be computationally expensive and take minutes, hours or even days to run. For tasks like optimisation, sensitivity analysis or uncertainty quantification where thousands or even millions of runs are needed, a solution has long been to approximate simulations with efficient emulators, which can be orders of magnitudes faster ([Forrester & Keane, 2009](#); [Kudela & Matousek, 2022](#)). Emulation is becoming increasingly widespread, ranging from engineering ([Yondo et al., 2018](#)), architecture ([Westermann & Evins, 2019](#)), biomedical ([Strocchi et al., 2023](#)) and climate science ([Bounceur et al., 2015](#)), to agent-based models ([Angione et al., 2022](#)).

A typical emulation pipeline involves three steps: 1. Evaluating the simulation at a small, strategically chosen set of inputs using techniques such as Latin Hypercube Sampling ([McKay et al., 1979](#)) to create a representative dataset, 2. constructing a high-accuracy emulator using that dataset, which involves model selection, hyperparameter optimisation and evaluation and 3. applying the emulator to tasks such as prediction, sensitivity analysis, or optimisation. A key challenge is the emulator construction, which requires machine learning expertise and familiarity with an evolving ecosystem of models and tools - creating a significant barrier for researchers focused on studying the underlying system rather than building emulators.

AutoEmulate automates emulator building, with the goal to eventually streamline the whole emulation pipeline. For people new to ML, AutoEmulate compares, optimises and evaluates a range of models to create an efficient emulator for their simulation in just a few lines of code. For experienced surrogate modellers, AutoEmulate provides a reference set of cutting-edge

emulators to quickly benchmark new models against. The package includes classic emulators such as Radial Basis Functions and Gaussian Processes, established ML models like Gradient Boosting and Support Vector Machines, as well as experimental deep learning emulators such as [Conditional Neural Processes](#) (Garnelo et al., 2018). AutoEmulate is built to be extensible. Emulators follow the popular [scikit-learn estimator template](#) and PyTorch (Paszke et al., 2019) deep learning models are supported with little overhead using a [skorch](#) (Tietz et al., 2017) interface.

AutoEmulate fills a gap in the current landscape of surrogate modeling tools as it's both highly accessible for newcomers while providing cutting-edge emulators for experienced surrogate modelers. In contrast, existing libraries either focus on lower level implementations of specific models, like GPflow (Matthews et al., 2017) and GPytorch (Gardner et al., 2018), or provide multiple emulators and applications but require to manually pre-process data, compare emulators and optimise hyperparameters like SMT in Python (Saves et al., 2024) or [Surrogates.jl](#) in Julia.

Pipeline

The inputs for AutoEmulate are X and y , where X is a 2D array (e.g. numpy-array, Pandas DataFrame) containing simulation parameters in columns and their values in rows, and y is an array containing the corresponding simulation outputs. A dataset X, y is usually obtained by constructing a set of parameters X using sampling techniques like Latin Hypercube Sampling (McKay et al., 1979) and evaluating the simulation on these inputs to obtain outputs y . With X and y , we can create an emulator with AutoEmulate in just a few lines of code.

```
from autoemulate.compare import AutoEmulate

ae = AutoEmulate()
ae.setup(X, y) # customise pipeline
ae.compare() # runs the pipeline

Under the hood, AutoEmulate runs a complete ML pipeline. It splits the data into training
and test sets, standardises inputs, fits a set of user-specified emulators, compares them using
cross-validation and optionally optimises hyperparameters using pre-defined search spaces.
All these steps can be customised in setup(). After running compare(), the cross-validation
results can be visualised and summarised.

ae.plot_cv() # visualise results
ae.summarise_cv() # cv scores for each model
```

Table 1: Average cross-validation scores

Model	Short Name	RMSE	R ²
Gaussian Process	gp	0.1027	0.9851
Random Forest	rf	0.1511	0.9677
Gradient Boosting	gb	0.1566	0.9642
Conditional Neural Process	cnp	0.1915	0.9465
Radial Basis Functions	rbf	0.3518	0.7670
Support Vector Machines	svm	0.4924	0.6635
LightGBM	lgbm	0.6044	0.4930
Second Order Polynomial	sop	0.8378	0.0297

After comparing cross-validation metrics and plots, an emulator can be selected and evaluated on the held-out test set (defaults to 20% of the data).

```
emulator = ae.get_model("GaussianProcess") # get fitted emulator
ae.evaluate(emulator)                     # calculate test set scores
ae.plot_eval(emulator, input_index=[0, 1]) # plot predictions
```

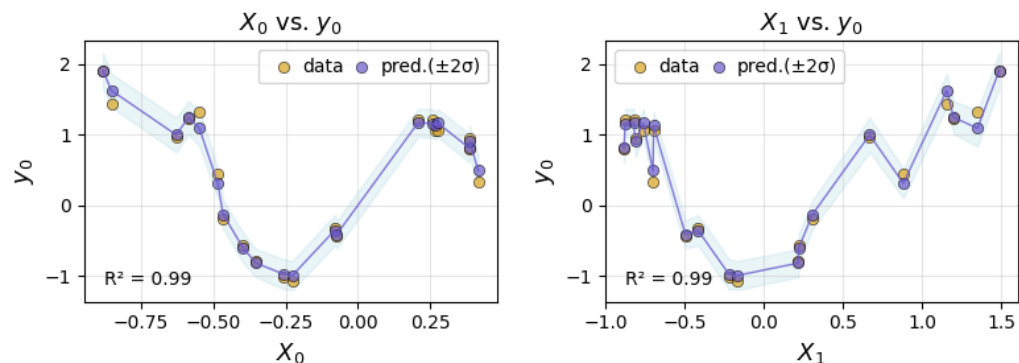


Figure 1: Test set predictions

Finally, the emulator can be refitted on the combined training and test set data before applying it. It's now ready to be used as an efficient replacement for the original simulation, being able to generate tens of thousands of new data points in negligible time using `predict()`. Lastly, we implemented global sensitivity analysis, which requires a large number of samples from the emulator to quantify how each simulation parameter and their interactions contribute to the output variance.

```
emulator = ae.refit(emulator)           # refit using full data
emulator.predict(X)                     # emulate!
ae.sensitivity_analysis(emulator)        # global SA with Sobol indices
```

Acknowledgements

We thank the Turing Research and Innovation Cluster in Digital Twins for supporting this work. We also thank Keith Worden, Ieva Kazlauskaitė, Rosie Williams, Robert Arthern, Peter Yatsyshin, Christopher Burr and James Byrne for discussions and Marina Strocchi, Zack Xuereb Conti, Richard Wilkinson for discussions and providing datasets.

References

- Angione, C., Silverman, E., & Yaneske, E. (2022). Using machine learning as a surrogate model for agent-based simulations. *PLOS ONE*, 17(2), e0263150. <https://doi.org/10.1371/journal.pone.0263150>
- Bounceur, N., Crucifix, M., & Wilkinson, R. D. (2015). Global sensitivity analysis of the climate–vegetation system to astronomical forcing: An emulator-based approach. *Earth System Dynamics*, 6(1), 205–224. <https://doi.org/10.5194/esd-6-205-2015>
- Forrester, A. I. J., & Keane, A. J. (2009). Recent advances in surrogate-based optimization. *Progress in Aerospace Sciences*, 45(1), 50–79. <https://doi.org/10.1016/j.paerosci.2008.11.001>
- Gardner, J., Pleiss, G., Weinberger, K. Q., Bindel, D., & Wilson, A. G. (2018). Gpytorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration. *Advances in Neural Information Processing Systems*, 31. <https://proceedings.neurips.cc/paper/2018/hash/27e8e17134dd7083b050476733207ea1-Abstract.html>

- Garnelo, M., Rosenbaum, D., Maddison, C. J., Ramalho, T., Saxton, D., Shanahan, M., Teh, Y. W., Rezende, D. J., & Eslami, S. M. A. (2018). *Conditional Neural Processes*. arXiv. <http://arxiv.org/abs/1807.01613>
- Kudela, J., & Matousek, R. (2022). Recent advances and applications of surrogate models for finite element method computations: A review. *Soft Computing*, 26(24), 13709–13733. <https://doi.org/10.1007/s00500-022-07362-8>
- Matthews, A. G. de G., Van Der Wilk, M., Nickson, T., Fujii, K., Boukouvalas, A., Le, P., Ghahramani, Z., & Hensman, J. (2017). GPflow: A Gaussian process library using TensorFlow. *Journal of Machine Learning Research*, 18(40), 1–6. <https://www.jmlr.org/papers/v18/16-537.html>
- McKay, M. D., Beckman, R. J., & Conover, W. J. (1979). A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. *Technometrics*, 21(2), 239–245. <https://doi.org/10.2307/1268522>
- Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., & Antiga, L. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32. <https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html>
- Saves, P., Lafage, R., Bartoli, N., Diouane, Y., Bussemaker, J., Lefebvre, T., Hwang, J. T., Morlier, J., & Martins, J. R. R. A. (2024). SMT 2.0: A Surrogate Modeling Toolbox with a focus on hierarchical and mixed variables Gaussian processes. *Advances in Engineering Software*, 188, 103571. <https://doi.org/10.1016/j.advengsoft.2023.103571>
- Strocchi, M., Longobardi, S., Augustin, C. M., Gsell, M. A. F., Petras, A., Rinaldi, C. A., Vigmond, E. J., Plank, G., Oates, C. J., Wilkinson, R. D., & Niederer, S. A. (2023). Cell to whole organ global sensitivity analysis on a four-chamber heart electromechanics model using Gaussian processes emulators. *PLOS Computational Biology*, 19(6), e1011257. <https://doi.org/10.1371/journal.pcbi.1011257>
- Tietz, M., Fan, T. J., Nouri, D., Bossan, B., & skorch Developers. (2017). *Skorch: A scikit-learn compatible neural network library that wraps PyTorch*. <https://skorch.readthedocs.io/en/stable/>
- Westermann, P., & Evins, R. (2019). Surrogate modelling for sustainable building design – A review. *Energy and Buildings*, 198, 170–186. <https://doi.org/10.1016/j.enbuild.2019.05.057>
- Yondo, R., Andrés, E., & Valero, E. (2018). A review on design of experiments and surrogate models in aircraft real-time and many-query aerodynamic analyses. *Progress in Aerospace Sciences*, 96, 23–61. <https://doi.org/10.1016/j.paerosci.2017.11.003>