# swift-emulator: A Python package for emulation of simulated scaling relations

**Roi Kugel**[*1] **and Josh Borrow**[†2]

**1** Leiden Observatory, Leiden University, PO Box 9513, NL-2300 RA Leiden, The Netherlands **2** Department of Physics, Kavli Institute for Astrophysics and Space Research, Massachusetts Institute of Technology, Cambridge, MA 02139, USA

**DOI:** 10.21105/joss.04240

**Software**

- Review
- Repository
- Archive

**Editor:** Dan Foreman-Mackey

**Reviewers:**

- @JDonaldM
- @kstoreyf

**Submitted:** 28 January 2022
**Published:** 11 April 2022

## Summary

`swift-emulator` is a Python toolkit for using Gaussian processes machine learning to emulate scaling relations from cosmological simulations. `swift-emulator` focusses on implementing a clear, easy to use design and API to remove the barrier to entry for using emulator techniques. `swift-emulator` provides tools for every step: the design of the parameter sampling, the training of the Gaussian process model, and validating and anaylsing the trained emulators. By making these techniques easier to use, in particular in combination with the SWIFT code (Borrow & Borrisov, 2020; Schaller et al., 2018), it will be possible use fitting methods (like MCMC) to calibrate and better understand theoretical simulation models.

## Statement of need

One of the limits of doing cosmological (hydrodynamical) simulations is that any simulation is limited to only a single set of parameters, be these choices of cosmology, or the implemented physics (e.g. stellar feedback). These parameters need to be tuned to calibrate against observational data. At odds with this, cosmological simulations are computationally expensive, with the cheapest viable runs costing thousands of CPU hours, and running up to tens of millions for the largest volumes at the highest resolutions. This makes the use of cosmological simulations in state-of-the-art fitting pipelines (e.g. MCMC), where tens of thousands to millions of evaluations of the model are required to explore the parameter space, computationally unfeasable. In order to get a statistical grip on the models of cosmology and galaxy formation, a better solution is needed.

This problem is a major limiting factor in "calibration" of the sub-resolution (subgrid) models that are often used. Works like Illustris (Vogelsberger et al., 2014), EAGLE (Crain et al., 2015), BAHAMAS (McCarthy et al., 2017), and Illustris-TNG (Pillepich et al., 2018) are able to "match" observed relations by eye, but a statistical ground for the chosen parameters is missing. This poses a signifcant problem for cosmology, where a deeper understanding of our subgrid models will be required to interpret results from upcoming surveys like LSST and EUCLID.

A solution here comes trough the use of machine learning techniques. Training 'emulators' on a limited amount of simulations enables the evaluation of a fully continuous model based on changes in the underlying parameters. Instead of performing a new simulation for each required datapoint, the emulator can predict the results a simulation would give for that set of parameters. This makes it feasable to use methods like MCMC based purely on simulation results.

[*] co-first author
[†] co-first author

## Emulator Requirements

For emulation in hydro simulations we want to use Gaussian processes to emulate scaling relations in the following form:

$$GP(y, x, \vec{\theta}).$$

We want to emulate scaling relations between a dependent variable $y$, as a function of the independent variable $x$ and the model parameters $\vec{\theta}$. For each simulation many of these individual scaling relations can be calculated, for example the sizes of galaxies relative to their stellar mass, or the mass fraction of gas in galaxy clusters as a function of their mass. The individual object properties used in scaling realtions can be measured from each individual simulation using a tool like VELOCIraptor (Elahi et al., 2019).

Between simulations, the underlying parameters $\vec{\theta}$ can change, for instance the energy injected by each supernovae. Using an emulator, we want to be able to see how many scaling relations change as a function of these parameters like the supernova strength.

Emulators do not make a distinction between the independent $x$ and the model parameters $\vec{\theta}$. An emulator will model $y$ as a function of the combined vector $\vec{\theta}' = (x, \vec{\theta})$. Getting the training data in the correct format can pose a significant challenge.

In order to save computational time, it is important to have an efficient sampling of the parameter space represented by $\vec{\theta}$. It may be more efficient to search the parameter space in a transformed coordinate space, like logarithmic space, if the expected viable range is over several orders of magnitude.

Once the emulator is working it can be challenging to perform standard tests to validate it. Things like cross-checks or parameter sweeps have to be implemented by hand, making proper use of emulators more difficult.

## Why `swift-emulator`?

Many packages exist for Gausian process emulation, like `george` (Ambikasaran et al. (2015); this provides the basis for `swift-emulator`), `gpytorch` (Gardner et al., 2018) and `GPy` (GPy, since 2012). Additionally, a package like `pyDOE` (Baudin et al., 2012) can be used to set up efficient parameter samplings. However, most of these packages operate close to theory, and create a significant barrier for entry.

With `swift-emulator` we aim to provide a single python package that interfaces with available tools at a high level. Additionaly we aim to streamline the processes by providing i/o tools for the SWIFT simulation code (Borrow & Borrisov, 2020; Schaller et al., 2018). This is done in a modular fashion, giving the users the freedom to change any steps along the way. `swift-emulator` provides many methods that work out of the box, removing the barrier to entry, and aim at making emulator methods easy to use. The more wide-spread use of emulators will boost the potential of future simulation projects.

`swift-emulator` combines these tools to streamline the complete emulation process. There are tools for experimental design, such as producing latin hypercubes or uniform samplings of $n$-dimensional spaces. For simulations performed with SWIFT, parameter files can be created and simulation outputs can be loaded in through helper methods in the library. The results can then be used to train an emulator that can make predictions for the scaling relations in the simulation. There are also methods to perform cross-checks to find the accuracy of the emulator. In addition, for investigating the impact of individual parameters on a given scaling relation, there is a simple method to do a parameter sweep implemented. Finally, there are tools for comparing the emulated relations with other data, from a simple $\chi^2$ method to complex model discrepancy structures.

`swift-emulator` is currently being used for two of the flagship simulation projects using the SWIFT simulation code, ranging across five orders of magnitude in mass resolution. The package is being used to allow modern simulations to reporduce key observations with high accuracy.

Finally `swift-emulator` has many options to optimise the methods for specific emulation problems. While the focus so far has been on integration with SWIFT, the underlying API is structured in a simple enough way that using the emulator with a different simulation code is easy. `swift-emulator` is currently being used for simulation projects outside of the SWIFT project for the calibration of postprocessing models.

# Acknowledgements

We acknowledge support from the SWIFT collaboration whilst developing this project, with notable involvement from Richard Bower, Ian Vernon, Joop Schaye, and Matthieu Schaller. This work is partly funded by Vici grant 639.043.409 from the Dutch Research Council (NWO). This work used the DiRAC@Durham facility managed by the Institute for Computational Cosmology on behalf of the STFC DiRAC HPC Facility (www.dirac.ac.uk). The equipment was funded by BEIS capital funding via STFC capital grants ST/K00042X/1, ST/P002293/1, ST/R002371/1 and ST/S002502/1, Durham University and STFC operations grant ST/R000832/1. DiRAC is part of the National e-Infrastructure.

# References

Ambikasaran, S., Foreman-Mackey, D., Greengard, L., Hogg, D. W., & O'Neil, M. (2015). Fast Direct Methods for Gaussian Processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *38*, 252. https://doi.org/10.1109/TPAMI.2015.2448083

Baudin, M., Christopoulou, M., Collette, Y., & Martinez, J.-M. (2012). pyDOE: The experimental design package for Python. In *GitHub repository*. GitHub. https://github.com/tisimst/pyDOE

Borrow, J., & Borrisov, A. (2020). swiftsimio: A Python library for reading SWIFT data. *The Journal of Open Source Software*, *5*(52), 2430. https://doi.org/10.21105/joss.02430

Crain, R. A., Schaye, J., Bower, R. G., Furlong, M., Schaller, M., Theuns, T., Dalla Vecchia, C., Frenk, C. S., McCarthy, I. G., Helly, J. C., Jenkins, A., Rosas-Guevara, Y. M., White, S. D. M., & Trayford, J. W. (2015). The EAGLE simulations of galaxy formation: calibration of subgrid physics and model variations. *Monthly Notices of the Royal Astronomical Society*, *450*(2), 1937–1961. https://doi.org/10.1093/mnras/stv725

Elahi, P. J., Poulton, R., & Canas, R. (2019). VELOCIraptor-STF: Six-dimensional Friends-of-Friends phase space halo finder. *The Astrophysics Source Code Library*, ascl:1911.020. http://ascl.net/1911.020

Gardner, J., Pleiss, G., Weinberger, K. Q., Bindel, D., & Wilson, A. G. (2018). GPyTorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 31). Curran Associates, Inc. https://proceedings.neurips.cc/paper/2018/file/27e8e17134dd7083b050476733207ea1-Paper.pdf

GPy. (since 2012). *GPy: A Gaussian process framework in Python*. http://github.com/SheffieldML/GPy.

McCarthy, I. G., Schaye, J., Bird, S., & Le Brun, A. M. C. (2017). The BAHAMAS project: calibrated hydrodynamical simulations for large-scale structure cosmology. *Monthly Notices of the Royal Astronomical Society*, *465*(3), 2936–2965. https://doi.org/10.1093/mnras/stw2792

Pillepich, A., Springel, V., Nelson, D., Genel, S., Naiman, J., Pakmor, R., Hernquist, L., Torrey, P., Vogelsberger, M., Weinberger, R., & Marinacci, F. (2018). Simulating galaxy formation with the IllustrisTNG model. *Monthly Notices of the Royal Astronomical Society*, *473*(3), 4077–4106. https://doi.org/10.1093/mnras/stx2656

Schaller, M., Gonnet, P., Draper, P. W., Chalk, A. B. G., Bower, R. G., Willis, J., & Hausammann, L. (2018). SWIFT: SPH With Inter-dependent Fine-grained Tasking. *The Astrophysics Source Code Library*, ascl:1805.020. http://ascl.net/1805.020

Vogelsberger, M., Genel, S., Springel, V., Torrey, P., Sijacki, D., Xu, D., Snyder, G., Nelson, D., & Hernquist, L. (2014). Introducing the Illustris Project: simulating the coevolution of dark and visible matter in the Universe. *Monthly Notices of the Royal Astronomical Society*, *444*(2), 1518–1547. https://doi.org/10.1093/mnras/stu1536