`hetGPy`: Heteroskedastic Gaussian Process Modeling in Python

David O'Gara [1,¶], **Mickaël Binois** [2], **Roman Garnett** [1,3], and **Ross A Hammond** [1,4,5,6]

1 Division of Computationl and Data Sciences, Washington University in St. Louis, USA **2** Acumes team, Université Côte d'Azur, Inria, Sophia Antipolis, France **3** Department of Computer Science, McKelvey School of Engineering, Washington University in St. Louis, USA **4** School of Public Health, Washington University in St. Louis, USA **5** Center on Social Dynamics and Policy, Brookings Institution, Washington DC, USA **6** Santa Fe Institute, Santa Fe, USA **¶** Corresponding author

Summary

Computer experiments are ubiquitous in the physical and social sciences. When experiments are time-consuming to run, emulator (or surrogate) models are often used to map the input–output response surface of the simulator, treating it as a black box. The workhorse function of emulator models is Gaussian process regression (GPR). GPs provide flexible, non-linear regression targets with good interpolation properties and uncertainty quantification. However, it is well-known that naïve GPR scales cubically with input size, and specifically involves intensive computation of matrix determinants and solving linear systems when fitting hyperparameters (Garnett, 2023; Gramacy, 2020). Further, naïve GPR with noisy observations typically assumes an independent, identically-distributed noise process, but many data-generating mechanisms, especially those found in stochastic computer simulation, exhibit non-equal variance noise (also known as heteroskedasticity) (Baker et al., 2020). The software package hetGP (Binois & Gramacy, 2021) alleviates both of these concerns when the dataset of interest contains multiple observations per location (i.e. a dataset of size N has $n << N$ unique input locations). The `hetGP` framework can calculate matrix inverses and determinants (and therefore the log likelihood and its gradients) on the order $O(n^3)$ rather than $O(N^3)$ time in the naïve case (Binois et al., 2018). This is accomplished via multiple applications of the Woodbury matrix identity (Binois et al., 2018; Harville, 1997) and leveraging the replication structure. Further, `hetGP` also models the mean and noise processes as two coupled GPs, allowing smooth noise dynamics over parameter space (Binois et al., 2018; Gramacy, 2020, 2020). The package has been used in a variety of contexts, such as statistics (Binois et al., 2018, 2019), biology (Lazaridis, 2022) and computational epidemiology (Shattock et al., 2022). We present a Python reimplementation `hetGPy`, developed in part due to Python's widespread use in computer simulation (Downey, 2023; Kinser, 2022).

Statement of Need

Python is a popular language for software development, data science, and computer experimentation. Its object oriented framework, high-level functionality, and third-party libraries make it a powerful tool for academic and industry professionals alike. Python is especially popular for computer simulation, with one particular example being the widespread use of Python models of COVID-19 spread (Aylett-Bullock et al., 2021; Kerr, 2021; O'Gara et al., 2023). `hetGPy` is well-posed for sequential design of Python models, mirroring the functionality of hetGP without having to rely on intermediate libraries such as `reticulate` (Ushey et al., 2023) or `rpy2` (Gautier, 2008), or in a more laborious case, converting a Python simulation to R.

The state of the art for GPR in Python is `GPyTorch` ([Gardner et al., 2018](#)), facilitated by black-box matrix–matrix multiplication (BBMM) which is extremely computationally efficient on GPUs. Other GPR routines for Python exist as well and can be found in libraries such as `PyMC` ([Abril-Pla et al., 2023](#)), `GPflow` ([Matthews et al., 2017](#)), and `GPJax` ([Pinder & Dodd, 2022](#)). However, while it is possible to jointly model the mean and variance as coupled GPs with these libraries, they do not take advantage of replication in datasets, meaning that under large degrees of replication and input-dependent noise, as is common in stochastic computer experiments ([Baker et al., 2020](#)), `hetGPy` will be more computationally efficient. [Figure 1](#) (a) shows a heteroskedastic GP fit to a simulated motorcycle accident dataset ([Silverman, 1985](#)). While it is possible to model input-dependent noise in `GPyTorch` or `BoTorch` ([Balandat et al., 2020](#)), specifying a smooth noise process in the method of ([Binois et al., 2018](#)) would require a custom implementation. [Figure 1](#) (b) illustrates the results of a simple one-dimensional example where we compare the model fits using both `hetGPy` and `GPyTorch`. While both models result in similar predictions, `hetGPy` is far faster, performing exact inference in less than 1 second, while GPyTorch takes over 10 seconds.
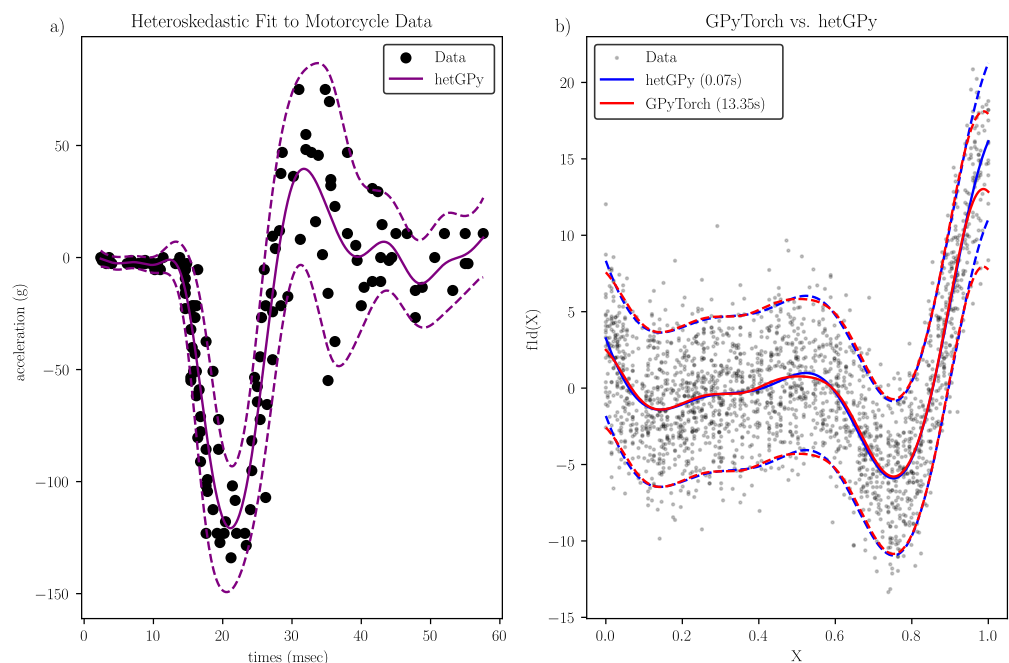
Figure 1: The main features of `hetGPy`. Panel (a) shows a heteroskedastic fit to the motorcycle data ([Silverman, 1985](#)). Panel (b) shows that `hetGPy` yields faster training than (naïve) `GPyTorch` with similar performance under high replication. Bolded and dashed lines indicate the predictive mean and 90% predictive intervals for homoskedastic GPR, respectively. Data were sampled from the f1d function $f(x) = (6x - 2)^2 \sin(12x - 4)$ ([Forrester et al., 2008](#)) in `hetGP` with between 1 and 20 replicates at each design location. Model training times in seconds are next to legend labels.

Additional Features and Case Study

hetGPy also has two intermediate goals: (1) efficient computations, accomplished via implementation on numpy arrays and (2) minimal dependencies, the core of which are `NumPy` ([Harris, 2020](#)) for efficient array-based computation and `SciPy` ([Virtanen, 2020](#)) which contains the definitive implementation of the L-BFGS-B algorithm ([Byrd et al., 1995](#); [Morales & Nocedal, 2011](#)) used for maximum likelihood estimation of hyperparameters. Our experiments indicate hetGPy is able to learn response surfaces efficiently, and in the case of high replication, do so on CPUs more efficiently than the default implementation in `GPyTorch`, as shown for a suite

of test problems in Table 1. The test problems contain $1,000$ unique design locations and an average of 5 replications per location, meaning $n = 1,000$ and $N \approx 5,000$ which shows why `hetGPy` is far more efficient in this setting. As a comparator, we also conduct a set of experiments using stochastic kriging (SK) ([Ankenman et al., 2010](#)), a precursor method to Binois et al. ([2018](#)) that also allows for maximum likelihood estimation with replication, and under the case of homoskedasticity, is nearly equivalent to homoscedastic GPR in `hetGP` and `hetGPy` ([Gramacy, 2020](#)). We implement SK with a custom `GPyTorch` likelihood that accounts for replication under homoskedasticity. Specifically, given a dataset X with unique designs $(X_1, ..., X_k)$ each replicated $(n_1, ..., n_k)$ times, we pre-average the outputs Y_i at each unique input location, and then estimate the diagonal of the noise matrix as (σ^2 / n_i). We see that for the SK case, `hetGPy` and `GPyTorch` have training times on a similar order of magnitude. The package `hetGPy` is under active development and is well-posed to engage with the wider Python community for future extension such as arbitrary kernel functions with auto-differentiation methods facilitated by `PyTorch` or `jax` ([Bradbury et al., 2018](#)).

Covariance	Experiment	Time (Seconds)			Number of Function Evaluations		
		GPyTorch (naïve)	GPyTorch (SK)	hetGPy	GPyTorch (naïve)	GPyTorch (SK)	hetGPy
Gaussian	Branin	98.45	5.61	2.96	13	18	13
	Goldstein-Price	119.02	3.7	2.04	21	12	10
	Hartmann-4D	98.57	4.67	5.15	18	15	16
	Hartmann-6D	466.38	21.05	21.49	95	69	40
	Sphere-6D	206.05	14.54	7.83	42	51	20
Matern $\nu = 5/2$	Branin	90.5	4.88	6.62	13	12	20
	Goldstein-Price	153.03	4.55	4.0	20	11	15
	Hartmann-4D	201.21	7.24	6.73	21	18	16
	Hartmann-6D	339.39	29.45	45.78	53	67	57
	Sphere-6D	147.12	16.89	16.46	24	38	14

Table 1: Comparing training times across a suite of test problems and libraries. All experiments reflect exact GPR with homoscedastic noise. Optimization problems were selected from ([Picheny et al., 2013](#)) with implementations from ([Surjanovic & Bingham, 2013](#)). Experiments consisted of a Latin Hypercube design of 1,000 unique locations, with between 1 and 10 replicates. iid Gaussian noise was added to each resulting dataset. Experiments were conducted on a 2019 MacBook Pro with a 2.6 GHz 6-Core Intel Core i7 processor, 16GB RAM on macOS Sonoma 14.7.1.

References

Abril-Pla, O., Andreani, V., Carroll, C., Dong, L., Fonnesbeck, C. J., Kochurov, M., Kumar, R., Lao, J., Luhmann, C. C., Martin, O. A., Osthege, M., Vieira, R., Wiecki, T., & Zinkov, R. (2023). PyMC: A modern, and comprehensive probabilistic programming framework in Python. *PeerJ Computer Science*, *9*, e1516. https://doi.org/10.7717/peerj-cs.1516

Ankenman, B., Nelson, B. L., & Staum, J. (2010). Stochastic Kriging for Simulation Metamodeling. *Operations Research*, *58*(2), 371–382. https://doi.org/10.1287/opre.1090.0754

Aylett-Bullock, J., Cuesta-Lazaro, C., Quera-Bofarull, A., Icaza-Lizaola, M., Sedgewick, A., Truong, H., Curran, A., Elliott, E., Caulfield, T., Fong, K., Vernon, I., Williams, J., Bower, R., & Krauss, F. (2021). JUNE: Open-source individual-based epidemiology simulation. *Royal Society Open Science*, *8*(7), 210506. https://doi.org/10.1098/rsos.210506

Baker, E., Barbillon, P., Fadikar, A., Gramacy, R. B., Herbei, R., Higdon, D., Huang, J., Johnson, L. R., Ma, P., Mondal, A., Pires, B., Sacks, J., & Sokolov, V. (2020). *Analyzing Stochastic Computer Models: A Review with Opportunities*. arXiv. https://doi.org/10.48550/arXiv.2002.01321

Balandat, M., Karrer, B., Jiang, D., Daulton, S., Letham, B., Wilson, A. G., & Bakshy, E. (2020). BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization. *Advances in Neural Information Processing Systems*, *33*, 21524–21538. https://proceedings.neurips.

cc/paper/2020/hash/f5b1b89d98b7286673128a5fb112cb9a-Abstract.html

Binois, M., & Gramacy, R. B. (2021). **hetGP** : Heteroskedastic Gaussian Process Modeling and Sequential Design in R. *Journal of Statistical Software*, *98*(13). https://doi.org/10.18637/jss.v098.i13

Binois, M., Gramacy, R. B., & Ludkovski, M. (2018). Practical Heteroscedastic Gaussian Process Modeling for Large Simulation Experiments. *Journal of Computational and Graphical Statistics*, *27*(4), 808–821. https://doi.org/10.1080/10618600.2018.1458625

Binois, M., Huang, J., Gramacy, R. B., & Ludkovski, M. (2019). Replication or Exploration? Sequential Design for Stochastic Simulation Experiments. *Technometrics*, *61*(1), 7–23. https://doi.org/10.1080/00401706.2018.1469433

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., & Zhang, Q. (2018). *JAX: Composable transformations of Python+NumPy programs* (Version 0.3.13). http://github.com/jax-ml/jax

Byrd, R. H., Lu, P., Nocedal, J., & Zhu, C. (1995). A Limited Memory Algorithm for Bound Constrained Optimization. *SIAM J. Sci. Comput.*, *16*(5), 1190–1208. https://doi.org/10.1137/0916069

Downey, A. B. (2023). *Modeling and Simulation in Python: An Introduction for Scientists and Engineers*. No Starch Press. ISBN: 9781718502161

Forrester, A. I. J., Sóbester, A., & Keane, A. J. (2008). *Engineering Design via Surrogate Modelling: A Practical Guide* (1st ed.). Wiley. https://doi.org/10.1002/9780470770801

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

Garnett, R. (2023). *Bayesian Optimization*. Cambridge University Press. ISBN: 9781108425780

Gautier, L. (2008). *rpy2: A simple and efficient access to R from Python*. https://github.com/rpy2/rpy2

Gramacy, R. B. (2020). *Surrogates: Gaussian Process Modeling, Design and Optimization for the Applied Sciences*. Chapman Hall/CRC. ISBN: 9780367415426

Harris, C. R. et al. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Harville, D. A. (1997). *Matrix Algebra From a Statistician's Perspective*. Springer. ISBN: 978-0-387-22677-4

Kerr, C. C. et al. (2021). Covasim: An agent-based model of COVID-19 dynamics and interventions. *PLOS Computational Biology*, *17*(7), e1009149. https://doi.org/10.1371/journal.pcbi.1009149

Kinser, J. M. (2022). *Modeling and Simulation in Python*. CRC Press. https://doi.org/10.1201/9781003226581

Lazaridis, I. et al. (2022). The genetic history of the Southern Arc: A bridge between West Asia and Europe. *Science*, *377*(6609), eabm4247. https://doi.org/10.1126/science.abm4247

Matthews, A. G. de G., Wilk, M. van der, Nickson, T., Fujii, K., Boukouvalas, A., León-Villagrá, P., Ghahramani, Z., & Hensman, J. (2017). GPflow: A Gaussian Process Library using TensorFlow. *Journal of Machine Learning Research*, *18*(40), 1–6. http://jmlr.org/papers/v18/16-537.html

Morales, J. L., & Nocedal, J. (2011). Remark on "algorithm 778: L-BFGS-B: Fortran subrou-

tines for large-scale bound constrained optimization." *ACM Transactions on Mathematical Software*, *38*(1), 7:1–7:4. https://doi.org/10.1145/2049662.2049669

O'Gara, D., Rosenblatt, S. F., Hébert-Dufresne, L., Purcell, R., Kasman, M., & Hammond, R. A. (2023). TRACE-Omicron: Policy Counterfactuals to Inform Mitigation of COVID-19 Spread in the United States. *Advanced Theory and Simulations*, 2300147. https://doi.org/10.1002/adts.202300147

Picheny, V., Wagner, T., & Ginsbourger, D. (2013). A benchmark of kriging-based infill criteria for noisy optimization. *Structural and Multidisciplinary Optimization*, *48*(3), 607–626. https://doi.org/10.1007/s00158-013-0919-4

Pinder, T., & Dodd, D. (2022). GPJax: A Gaussian Process Framework in JAX. *Journal of Open Source Software*, *7*(75), 4455. https://doi.org/10.21105/joss.04455

Shattock, A. J., Le Rutte, E. A., Dünner, R. P., Sen, S., Kelly, S. L., Chitnis, N., & Penny, M. A. (2022). Impact of vaccination and non-pharmaceutical interventions on SARS-CoV-2 dynamics in Switzerland. *Epidemics*, *38*, 100535. https://doi.org/10.1016/j.epidem.2021.100535

Silverman, B. W. (1985). Some Aspects of the Spline Smoothing Approach to Non-Parametric Regression Curve Fitting. *Journal of the Royal Statistical Society: Series B (Methodological)*, *47*(1), 1–21. https://doi.org/10.1111/j.2517-6161.1985.tb01327.x

Surjanovic, S., & Bingham, D. (2013). *Virtual Library of Simulation Experiments: Test Functions and Datasets*. https://www.sfu.ca/~ssurjano/

Ushey, K., Allaire, J. J., & Tang, Y. (2023). *Reticulate: Interface to 'Python'*. https://CRAN.R-project.org/package=reticulate

Virtanen, P. et al. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, *17*(3), 261–272. https://doi.org/10.1038/s41592-019-0686-2