The Journal of Open Source Software

hetGPy: Heteroskedastic Gaussian Process Modeling in Python

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

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

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor:

Submitted: 18 October 2024
Published: unpublished

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 (Garnett, 2023; Gramacy, 2020) when fitting hyperparameters. 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 input-dependent noise (also known as heteroskedasticity) (Baker et al., 2020). The software package `hetGP` (Binois & Gramacy, 2021) alleviates these both of these concerns: when the dataset of interest contains replicates, as it is possible to perform inference and prediction with cost growing cubically in the number of unique design locations n rather than the full dataset of size N and can jointly model the mean and noise process as two coupled GPs, allowing smooth noise dynamics over parameter space (Binois et al., 2018). The package has been used in a variety of contexts, such as statistics (Binois et al., 2018, 2019), biology (Lazaridis et al., 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 orientated framework, high-level functionality, and third-party libraries such as `numpy` (Harris et al., 2020), `scipy` (Virtanen et al., 2020), `pytorch` (Paszke et al., 2019) and `scikit-learn` (Pedregosa et al., 2011) 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 et al., 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`, 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, to our knowledge, these libraries, under their default behavior, do not jointly model the mean and variance as coupled GPs or take advantage of replication in datasets, meaning that under large degrees of replication and input-dependent noise, as in 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 several orders of magnitude faster, performing exact inference in less than 1 second, while GPyTorch takes over 10 seconds.

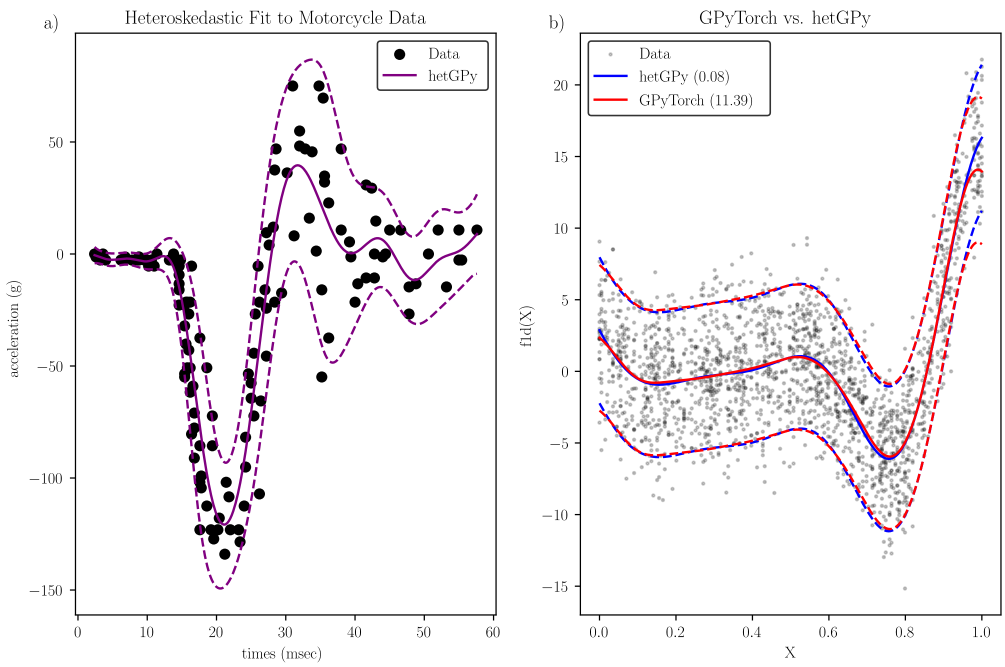

Figure 1: The main features of `hetGPy`. Panel (a) shows a heteroskedastic fit to the motorcycle data (Silverman, 1985). Panel (b) shows that `hetGPy` yields faster training than (naïve) `GPyTorch` with similar performance under high replication. Bolded and dashed lines indicate the predictive mean and 90% predictive intervals for homoscedastic 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.

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 for efficient array-based computation and scipy which contains the definitive implementation of the L-BFGS-B algorithm in Fortran (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. 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 $(\sigma^2 \ / \ n_i)$. We see that the 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`.

Covariance	Experiment	Time (seconds) GPyTorch (naïve)	GPyTorch (SK)	hetGPy	Number of Evaluations GPyTorch (naïve)	GPyTorch (SK)	hetGPy
Gaussian	Branin	98.449	5.622	2.693	13	18	13
	Goldstein-Price	119.019	3.688	1.895	21	12	10
	Hartmann-4D	98.566	4.578	4.936	18	15	16
	Hartmann-6D	466.381	19.845	21.472	95	69	40
	Sphere-6D	206.05	13.989	7.702	42	51	20
Matern $\nu = 5/2$	Branin	90.501	4.765	6.563	13	12	20
	Goldstein-Price	153.033	4.406	4.005	20	11	15
	Hartmann-4D	201.208	6.895	6.755	21	18	16
	Hartmann-6D	339.387	28.427	45.23	53	67	57
	Sphere-6D	147.118	16.034	16.345	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, n.d.). 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.

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). J une : 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

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.

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.

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

Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Kerr, C. C., Stuart, R. M., Mistry, D., Abeysuriya, R. G., Rosenfeld, K., Hart, G. R., Núñez, R. C., Cohen, J. A., Selvaraj, P., Hagedorn, B., George, L., Jastrzębski, M., Izzo, A. S., Fowler, G., Palmer, A., Delport, D., Scott, N., Kelly, S. L., Bennette, C. S., … Klein, D. J. (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.

Lazaridis, I., Alpaslan-Roodenberg, S., Acar, A., Açıkkol, A., Agelarakis, A., Aghikyan, L., Akyüz, U., Andreeva, D., Andrijašević, G., Antonović, D., Armit, I., Atmaca, A., Avetisyan, P., Aytek, A. İ., Bacvarov, K., Badalyan, R., Bakardzhiev, S., Balen, J., Bejko, L., … Reich, D. (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., Le{\'o}n-Villagr{\'a}, 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 subroutines 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

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., … Chintala, S. (2019). *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. arXiv. https://doi.org/10.48550/arXiv.1912.01703

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

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. (n.d.). *Virtual Library of Simulation Experiments: Test Functions and Datasets*.

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

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., Walt, S. J. van der, Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., … Mulbregt, P. van. (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