Journal of Machine Learning Research 19 (2018) 1-7 Submitted 7/17; Revised 5/18; Published 8/18

ELFI: Engine for Likelihood-Free Inference

Jarno Lintusaari JARNO.LINTUSAARI@AALTO.FI
Henri Vuollekoski HENRI.VUOLLEKOSKI@AALTO.FI
Antti Kangasrääsiö ANTTI.KANGASRAASIO@AALTO.FI
Kusti Skytén KUSTI.SKYTEN@AALTO.FI
Marko Järvenpää MARKO.J.JARVENPAA@AALTO.FI
Pekka Marttinen PEKKA.MARTTINEN@AALTO.FI
Department of Computer Science
Aalto University
00076 Aalto, Finland

Michael U. Gutmann MICHAEL.GUTMANN@ED.AC.UK
School of Informatics
The University of Edinburgh
Edinburgh, EH8 9AB, UK

Aki Vehtari * AKI.VEHTARI@AALTO.FI
Department of Computer Science
Aalto University
00076 Aalto, Finland

Jukka Corander * JUKKA.CORANDER@MEDISIN.UIO.NO
Department of Biostatistics
University of Oslo
0317 OSLO, Norway

Samuel Kaski * SAMUEL.KASKI@AALTO.FI
Department of Computer Science
Aalto University
00076 Aalto, Finland

Editor: Alexandre Gramfort

Abstract

Engine for Likelihood-Free Inference (ELFI) is a Python software library for performing likelihood-free inference (LFI). ELFI provides a convenient syntax for arranging components in LFI, such as priors, simulators, summaries or distances, to a network called ELFI graph. The components can be implemented in a wide variety of languages. The stand-alone ELFI graph can be used with any of the available inference methods without modifications. A central method implemented in ELFI is Bayesian Optimization for Likelihood-Free Inference (BOLFI), which has recently been shown to accelerate likelihood-free inference up to several orders of magnitude by surrogate-modelling the distance. ELFI also has an inbuilt support for output data storing for reuse and analysis, and supports parallelization of computation from multiple cores up to a cluster environment. ELFI is designed to be extensible and provides interfaces for widening its functionality. This makes the adding of new inference methods to ELFI straightforward and automatically compatible with the inbuilt features.

∗. Equal contribution

Keywords: Likelihood-free inference, approximate Bayesian computation, Python, BOLFI, parallel computing

1. Introduction

Engine for Likelihood-Free Inference (ELFI) is a statistical software package for likelihood-free inference written in Python. The term "likelihood-free inference" (LFI) refers to a family of inference methods that can be used when the likelihood function is not computable or otherwise available, but it is possible to simulate from the model (see e.g. Lintusaari et al., 2017). Other names for likelihood-free inference or closely related approaches include Approximate Bayesian Computation (ABC) (see e.g. Marin et al., 2012; Lintusaari et al., 2017), simulator-based inference, approximative Bayesian inference and indirect inference.

In LFI, generative models are commonly composed of priors and user-specified simulators. The inference is based on the outputs of the generative model, that is, on the simulated data for various parameter configurations, as opposed to the likelihoods of the observed data under the configurations. To facilitate the inference, the observed and simulated data are usually summarized after which distances between the summaries are taken. In ELFI, the simulators, summaries, distances, etc. are called components and can be implemented in a wide variety of languages.

One of the main features in ELFI is the convenient syntax of combining all of the components into a single network (Figure 1) that we call an ELFI graph. Once the ELFI graph is specified, it can be used with any of the available inference algorithms. ELFI also supports parallelization of the inference from a single computer up to a computational cluster, and storing the generated data for reuse, post-processing and further analysis. ELFI has emerged from the prior research on the subject by the authors (Lintusaari et al., 2016; Gutmann and Corander, 2016; Lintusaari et al., 2017; Kangasrääsiö et al., 2017) and was used by Kangasrääsiö et al. (2017) and Kangasrääsiö and Kaski (2017).

2. Software Design Principles

ELFI is designed to support likelihood-free inference research both from the practitioners' and methodologists' point of view. We aim for an easy-to-use ecosystem where practitioners will find the state-of-the-art inference methods, whereas methodologists will find simulators along with accompanying ELFI graphs and data to aid in method development and assessment.

2.1 Features for Practitioners

For practitioners ELFI provides a convenient interface for quickly arranging the components needed in LFI into an ELFI graph. Inherently ELFI graphs are directed acyclic graphs (DAGs), that represent how quantities used by the inference algorithm (e.g. distances) are computed (Figure 1). The DAG structure makes it possible to construct detailed hierarchies between the components (nodes). Under the hood, the ELFI graph is converted to a computation graph that will include e.g. nodes for the observed data (see the documentation[1] for more details). The nodes (components) are either data or operations that output data. Users are free to implement components as needed, but ELFI provides also ready made implementations for common components. Once specified, the ELFI graph can be directly

```python
# Define the simulator, the summary and the observed data
def simulator(t1, t2, batch_size=1, random_state=None):
    # Implementation comes here. Return `batch_size`
    # simulations wrapped to a NumPy array.
def summary(data, argument=0):
    # Implementation comes here...
y = # Observed data, as one element of a batch.

# Specify the ELFI graph
t1 = elfi.Prior('uniform', -2, 4)
t2 = elfi.Prior('normal', t1, 5)  # depends on t1
SIM = elfi.Simulator(simulator, t1, t2, observed=y)
S1 = elfi.Summary(summary, SIM)
S2 = elfi.Summary(summary, SIM, 2)
d = elfi.Distance('euclidean', S1, S2)

# Run the rejection sampler
rej = elfi.Rejection(d, batch_size=10000)
result = rej.sample(1000, threshold=0.1)
```

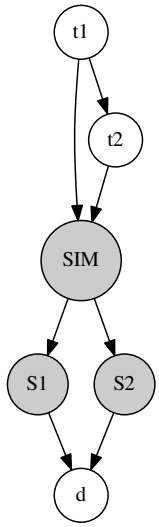

Figure 1: Example of an ELFI graph and of running ABC rejection sampling in ELFI. The observed data are given to the operation that produces corresponding output. Two summaries are defined, where summary S2 is given an additional argument 2.

used with any of the available inference methods. We have provided an initial set of methods that can handle different types of scenarios: basic rejection sampling for cheap simulators, the general-purpose sequential Monte Carlo as well as BOLFI (Gutmann and Corander, 2016) for expensive simulators. BOLFI combines probabilistic modelling of the distance with decision-making under uncertainty to decide for which parameter value to next run the simulator, significantly reducing the number of simulations needed.

Since likelihood-free inference often requires a moderate amount of experimentation (e.g. trying different summary statistics) it is important that specifying the components is made flexible and that already generated data can be reused. We found the DAG structure to be ideal for these tasks. First, ELFI allows any part of the ELFI graphs (e.g. nodes and their dependencies) to be redefined keeping the rest of the structure intact. Second, ELFI provides automatic storing of the full output of any node (e.g. the simulator). These data will be automatically reused when for example the summaries are changed, potentially resulting in significant savings in compute time. These features are demonstrated in the ELFI tutorial in the documentation.

Another important factor is the ability to use non-Python components in the ELFI graph. For instance, it may not always be practical or even possible to rewrite existing simulators in Python. ELFI provides both tools and examples in the documentation on how to use simulators written in other languages.

Other practical features include the ability to progress the inference iteratively and to stop early if necessary. The provided visualization functions support assessing the current state of the inference. Finally, the ELFI graph can be saved to a file and shared with

others. ELFI also guarantees that the results will be identical for the same seeds making the reproduction of the results easy.

2.2 Features for Methodologists

For methodologists ELFI provides a convenient platform for testing new algorithms with models from the literature (e.g. Ricker, 1954; Marin et al., 2012; Lintusaari et al., 2017) and comparing their performance against existing algorithms. The framework provides means for parallelization, data storing, seeding of pseudo random number generation and other important technicalities out of the box. The documentation includes instructions on how to implement new algorithms for ELFI. One of the major benefits is that all existing ELFI graphs will be usable with the new algorithms without modifications.

3. Performance and Scalability

Performance is an important factor in computationally heavy inference such as LFI. ELFI uses batches of computations to control execution performance and parallelization. A batch consists of a fixed number of consecutive evaluations of a node in the ELFI graph before moving to the next (e.g. 100 draws from the prior and then 100 simulations using those parameters). The standard parallelization strategy is to compute multiple batches in parallel. This provides several benefits. First, the computation of a single batch can often be vectorized with, for example, NumPy (van der Walt et al., 2011) for many of the basic operations (e.g. computing summaries or distances), making them efficient in Python. This is especially beneficial when experimenting with different summaries and distances with precomputed simulations. Batches are also often relatively constant in their time and memory consumption, allowing flexibility in planning the parallel execution of multiple batches. This helps in avoiding unnecessary message passing, progressing the inference in meaningful steps, and makes it possible to know in advance the size of the returned output data for storing purposes.

4. Comparison to Other Similar Software

There exist multiple LFI libraries for parameter inference. Many of them are either restricted to a specific problem domain (Liepe et al., 2014; Cornuet et al., 2014; Louppe et al., 2016), or require existing simulated data (Thornton, 2009; Csilléry et al., 2012; Nunes and Prangle, 2015). Edward (Tran et al., 2016) provides some LFI methods with a GPU acceleration, but requires the simulator to be differentiable. ELFI makes no extra requirements for the simulator (or other components), and can also be used with implementations taking benefit of hardware accelerations (e.g. GPU). General-purpose LFI software similar to ELFI are, to our knowledge, ABCtoolbox (Wegmann et al., 2010), EasyABC (Jabot et al., 2013), and ABCpy (Dutta et al., 2017). A relatively recent categorization of LFI software is provided by Nunes and Prangle (2015).

Among the general-purpose LFI software, only ELFI separates the LFI component specification from inference (Table 1). The graph-based specification provides considerable flexibility in both defining the components and experimenting with them. For example, it

Software	Language	Latest release	Data reuse	Parallelization	Graph-based	Iterative processing
ABCtoolbox	C++	2009	Partial	cluster (manual)	✗	✗
EasyABC	R	2015	Partial	local	✗	✗
ABCpy	Python	2017	✗	local and cluster	✗	✗
ELFI	Python	2017	✓	local and cluster	✓	✓

Table 1: Comparison of general-purpose LFI frameworks

is possible to embed multiple simulators into a single ELFI graph without modifying their codes. We refer the reader to the documentation for illustrations.[1]

Regarding parallelization, EasyABC supports multiple cores on a single computer while the others can also run in cluster environments. By default, ABCpy uses Spark (Zaharia et al., 2010) and ELFI ipyparallel for parallelization but both can be used with alternative backends.[2] ABCtoolbox does not provide a parallel solution out of the box.

ABCtoolbox, EasyABC and ELFI support reusing generated data. ELFI is more flexible in that it allows the output of any node of the ELFI graph to be stored, and it automatically uses that data to compute the output of its current or future child nodes. There is thus no need to manually transform existing data.

Only ELFI supports advancing the inference sample-by-sample, which facilitates debugging and enables e.g. convergence monitoring and early stopping. Also, ELFI is currently the only general-purpose software to implement the BOLFI method (Gutmann and Corander, 2016), which can handle expensive-to-evaluate simulators outside the reach of other methods.

5. Source Code and Dependencies

ELFI has been designed to be open source and modular, and can be extended through interfaces. For instance, it is possible to add new types of components, data stores or parallel clients. All the dependencies of ELFI are also open source.

ELFI is written in Python and is officially tested under Linux and MacOS but also works in Windows. The code style follows PEP 8 and documentation NumPy format. Code development uses the continuous integration practice with code review and automated tests to ensure the quality and usability of the software. The venue for distributing the source is GitHub that among the above features also allows anyone to raise issues regarding the software and make pull requests for new features.[3] Online documentation is hosted in the Read The Docs.[1] ELFI also has a community chat for the users.

Acknowledgments

We would like to acknowledge support for this project from the Academy of Finland (Finnish Centre of Excellence in Computational Inference Research COIN) and grants 294238, 292334. We acknowledge the computational resources provided by the Aalto Science-IT project.

1. ELFI documentation can be found at `http://elfi.readthedocs.io`.
2. The ipyparallel project can be found at `https://github.com/ipython/ipyparallel`.
3. The ELFI GitHub repository can be found at `https://github.com/elfi-dev/elfi`.

References

J. M. Cornuet, P. Pudlo, J. Veyssier, A. Dehne-Garcia, M. Gautier, R. Leblois, J. M. Marin, and A. Estoup. DIYABC v2.0: a software to make approximate Bayesian computation inferences about population history using single nucleotide polymorphism, DNA sequence and microsatellite data. *Bioinformatics*, 30(8):1187–1189, Apr 2014.

K. Csilléry, O. François, and M. G. B. Blum. abc: an R package for approximate Bayesian computation (ABC). *Methods in Ecology and Evolution*, 3(3):475–479, 2012. doi: 10.1111/j.2041-210X.2011.00179.x.

R. Dutta, M. Schoengens, J-P. Onnela, and A. Mira. ABCpy: A user-friendly, extensible, and parallel library for approximate Bayesian computation. In *Proceedings of the Platform for Advanced Scientific Computing Conference*, PASC '17, pages 8:1–8:9, New York, NY, USA, 2017. ACM. doi: 10.1145/3093172.3093233. URL `http://doi.acm.org/10.1145/3093172.3093233`.

M. U. Gutmann and J. Corander. Bayesian optimization for likelihood-free inference of simulator-based statistical models. *Journal of Machine Learning Research*, 17(125):1–47, 2016.

F. Jabot, T. Faure, and N. Dumoulin. EasyABC: performing efficient approximate Bayesian computation sampling schemes using R. *Methods in Ecology and Evolution*, 4(7):684–687, 2013. doi: 10.1111/2041-210X.12050.

A. Kangasrääsiö and S. Kaski. Inverse reinforcement learning from summary data. *arXiv preprint arXiv:1703.09700*, 2017.

A. Kangasrääsiö, K. Athukorala, A. Howes, J. Corander, S. Kaski, and A. Oulasvirta. Inferring cognitive models from data using approximate Bayesian computation. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, pages 1295–1306, New York, NY, USA, 2017. ACM. doi: 10.1145/3025453.3025576.

J. Liepe, P. Kirk, S. Filippi, T. Toni, C. P. Barnes, and M. P. Stumpf. A framework for parameter estimation and model selection from experimental data in systems biology using approximate Bayesian computation. *Nat Protoc*, 9(2):439–456, Feb 2014.

J. Lintusaari, M. U. Gutmann, S. Kaski, and J. Corander. On the identifiability of transmission dynamic models for infectious diseases. *Genetics*, 2016. doi: 10.1534/genetics.115.180034.

J. Lintusaari, M. U. Gutmann, R. Dutta, S. Kaski, and J. Corander. Fundamentals and recent developments in approximate Bayesian computation. *Systematic Biology*, 66(1):e66, 2017. doi: 10.1093/sysbio/syw077.

G. Louppe, K Cranmer, and J. Pavez. carl: a likelihood-free inference toolbox, March 2016. URL `http://dx.doi.org/10.5281/zenodo.47798`.

J-M. Marin, P. Pudlo, C. P. Robert, and R. J. Ryder. Approximate Bayesian computational methods. *Statistics and Computing*, 22(6):1167–1180, 2012. doi: 10.1007/s11222-011-9288-2.

M. A. Nunes and D. Prangle. abctools: An R Package for Tuning Approximate Bayesian Computation Analyses. *The R Journal*, 7(2):189–205, 2015.

W. E. Ricker. Stock and recruitment. *Journal of the Fisheries Research Board of Canada*, 11(5):559–623, 1954. doi: 10.1139/f54-039.

K. R. Thornton. Automating approximate Bayesian computation by local linear regression. *BMC Genetics*, 10(1):35, 2009. doi: 10.1186/1471-2156-10-35.

D. Tran, A. Kucukelbir, A. B. Dieng, M Rudolph, D. Liang, and D. M. Blei. Edward: A library for probabilistic modeling, inference, and criticism. *arXiv preprint arXiv:1610.09787*, 2016.

S. van der Walt, S. C. Colbert, and G. Varoquaux. The numpy array: A structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011. doi: 10.1109/MCSE.2011.37.

D. Wegmann, C. Leuenberger, S. Neuenschwander, and L. Excoffier. ABCtoolbox: a versatile toolkit for approximate Bayesian computations. *BMC Bioinformatics*, 11(1):116, 2010. doi: 10.1186/1471-2105-11-116.

M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association.