Statistical Theory and Related Fields

ISSN: 2475-4269 (Print) 2475-4277 (Online) Journal homepage: www.tandfonline.com/journals/tstf20

glabcmcmc: a Python package for ABC-MCMC with local and global moves

Xuefei Cao, Shijia Wang & Yongdao Zhou

To cite this article: Xuefei Cao, Shijia Wang & Yongdao Zhou (2025) glabcmcmc: a Python package for ABC-MCMC with local and global moves, Statistical Theory and Related Fields, 9:2, 168-177, DOI: 10.1080/24754269.2025.2495505

To link to this article: https://doi.org/10.1080/24754269.2025.2495505

Published online: 05 May 2025.

STATISTICAL THEORY AND RELATED FIELDS
2025, VOL. 9, NO. 2, 168–177
https://doi.org/10.1080/24754269.2025.2495505

glabcmcmc: a Python package for ABC-MCMC with local and global moves

Xuefei Cao[a], Shijia Wang [b] and Yongdao Zhou[a]

[a]NITFID, School of Statistics and Data Science, Nankai University, Tianjin, People's Republic of China; [b]Institute of Mathematical Sciences, ShanghaiTech University, Shanghai, People's Republic of China

ABSTRACT

We introduce a new Python package glabcmcmc, which implements an approximate Bayesian computation Markov chain Monte Carlo (ABC-MCMC) algorithm that combines global and local proposal strategies to address the limitations of standard ABC-MCMC. The proposed package includes key innovations such as the determination of global proposal frequencies, the implementation of a hybrid ABC-MCMC algorithm integrating global and local proposals, and an adaptive version that utilizes normalizing flows and gradient-based computations for enhanced proposal mechanisms. The functionality of the software package is demonstrated through illustrative examples.

ARTICLE HISTORY
Received 25 January 2025
Revised 2 April 2025
Accepted 15 April 2025

KEYWORDS
Approximate Bayesian Computation; Markov chain Monte Carlo; global-local proposal

1. Introduction

Approximate Bayesian Computation (ABC) (Beaumont et al., 2002; Pritchard et al., 1999) is a likelihood-free inference method for posterior approximation when the likelihood function is intractable but sampling from the model is possible.

Numerous software packages are available for ABC inference: the R package abc (Csilléry et al., 2012) implements the ABC rejection algorithm and provides various regression post-processing methods. The R package EasyABC (Easy, 2013) offers multiple ABC algorithms, including five sequential sampling schemes and three schemes coupled with MCMC. ABCtoolbox (Wegmann et al., 2010) runs on both Linux and Windows platforms and is specialized for ABC analysis of genetic models. In addition, EP−ABC (Barthelmé & Chopin, 2014) is a MATLAB toolbox for ABC analysis of state-space models and related models, and ABC−SDE (Picchini, 2014) is another MATLAB toolbox focussed on stochastic differential equations. ABC−SysBio (Liepe et al., 2010) includes Python scripts for ABC analysis in systems biology, and the R package abcrf (Raynal et al., 2019) is designed to implement ABC random forests for Bayesian parameter inference. The R package ejMCMC (Cao et al., 2024b) utilizes a Gaussian process model to early reject some candidate parameters, thereby accelerating ABC-MCMC inference. These packages or standalone software

CONTACT Shijia Wang wangshj1@shanghaitech.edu.cn, shijia_wang@nankai.edu.cn Institute of Mathematical Sciences, ShanghaiTech University, Shanghai, People's Republic of China; Yongdao Zhou ydzhou@nankai.edu.cn NITFID, School of Statistics and Data Science, Nankai University, Tianjin, People's Republic of China

tools are specifically designed for conducting ABC inference, targeting either a broad spectrum or particular types of model. They generally employ rejection sampling algorithms or MCMC moves with simple proposal distributions.

In contrast, our Python package introduces an innovative approach, utilizing ABC-MCMC with global-local proposal algorithms (GL-ABC-MCMC) described in Cao et al. (2024a). The package provides a unified framework for performing likelihood-free Bayesian inference by combining global and local proposal strategies to improve sampling efficiency, and can be applied to parameter inference across various models. It offers a variety of GL-ABC-MCMC sampling methods, including the usage of a distribution directly as a proposal, employing iterative sampling importance resampling (iSIR) to construct the global proposal, utilizing the gradient-based Metropolis-Adjusted Langevin algorithm (MALA) as a local proposal, and enhancing the global proposal with normalizing flows. Additionally, the package provides functionality to evaluate the expected square jump distance (ESJD), a criterion used to select appropriate hyperparameters.

2. Review of global-local ABC-MCMC

The basic ABC algorithm for posterior inference $p(\theta|y)$ is the rejection sampling. MCMC methods are commonly used in ABC inference; however, their inherent proposal mechanisms often cause them to get stuck in local regions, impacting the algorithm's convergence speed. This paper presents an implementation package for a more efficient algorithm, the GL-ABC-MCMC algorithm. In each iteration of the MCMC process, the algorithm selects a global proposal with a probability of γ , otherwise opting for a local proposal. This approach balances the exploration capability of the global proposal with the exploitation capability of the local proposal. When γ is set to 0 or 1, the algorithm reduces to the standard ABC-MCMC, making it more versatile. Regarding the local proposals, users can use a Gaussian proposal, or they can opt for an adaptive version, that is, the Metropolis-Adjusted Langevin algorithm (MALA) based on gradient calculations. For global proposals, there are two options: users can choose a specific distribution, or construct global proposals using iSIR. Additionally, normalizing flows can be employed to improve the global proposals.

3. Key features

You can install `glabcmcmc` using pip:

```
git clone https://github.com/caofff/GL-ABC-MCMC
cd GL-ABC-MCMC
pip install -e .
```

3.1. Core components

The package offers four implementations of the GL-ABC-MCMC algorithm.

(1) **GlobalMCMC:** This combines parametric global and local proposal distributions.

(2) **GLMCMC:** The global proposal of **GlobalMCMC** is replaced with one constructed using iSIR.
(3) **GLMALA:** This version uses MALA as the local proposal in the **GLMCMC** algorithm.
(4) **GLMCMC-NFs:** Building upon GLMCMC, this implementation utilizes normalizing flows to enhance the global proposal distribution.

And the **MCMCRunner** class provides a unified interface for all MCMC methods. We can define a 'runner' by inputting the ABC model set and the directory where posterior samples are stored.

```
runner = MCMCRunner(
    abc_set,          # Problem-specific ABC set
    output_dir='./' # Output directory and the default is
                      # the current directory.
)
```

Before conducting Bayesian inference, it is essential to define several model functions. These include the function for generating simulated data based on parameters, the discrepancy function, the kernel function, and the prior function. The implementation of these components can be effectively managed using the ABCset interface, structured as follows.

```
class ABCSet:
    def __init__(self, epsilon):
        self.epsilon = epsilon
        self.theta_dim = theta_dim
        self.y = y_obs
    def generate_samples(self, theta):
        """Generate synthetic samples for
        parameters theta."""
        pass
    def discrepancy(self, x):
        """Compute discrepancy between simulated and
        observed data."""
        pass
    def calculate_log_kernel(self, x):
        """Compute log of ABC kernel."""
        pass
    def prior_log_prob(self, theta):
        """Compute log prior probability."""
        pass
```

For precise implementation details, `Example/Mixabs.py` is recommended.

Then, we can invoke four different ABC-MCMC with global and local methods through 'runner.∗()'. The specific operations are as follows.

(1) **GlobalMCMC**:

```
runner.run_global_mcmc(
    num_iterations,    # Number of iterations
    initial_theta,     # Initial parameters
    initial_y,         # Initial synthetic data
    global_frequency,  # Global frequency
    local_proposal,    # Local proposal
    global_proposal,   # Global proposal
    output_file = 'global_mcmc_results.csv'
# Output file, default is 'global_mcmc_results.csv'.
)
```

(2) **GLMCMC**:

```
runner.run_glmcmc(num_iterations, initial_theta,
    initial_y, global_frequency, local_proposal,
    importance_proposal, # Importance proposal of iSIR
    batch_size,          # Batch_size of iSIR.
    output_file='glmcmc_results.csv'
)
```

(3) **GLMALA**:

```
runner.run_glmala(num_iterations, initial_theta,
    initial_y, global_frequency, importance_proposal,
    batch_size, tau, # Step-size of MALA
    num_grad,        # Simulation number used for
                     # calculating gradients.
    output_file='glmala_results.csv'
    )
```

(4) **GLMCMC-NF**:

```
runner.run_glmcmc_nf(num_iterations,  initial_theta,
    initial_y, global_frequency, local_proposal,
    batch_size, importance_proposal_base,
    # The initial importance proposal of iSIR
    step_size,
    #The frequency of updating the importance proposal.
    train_steps,
    #The total times of updating the importance proposal.
    output_file='glmcmc_nf_results.csv'
)
```

Proposal distributions like 'local_proposal', 'global_propoal', and 'importance_proposal' are classes that include the 'forward' function and the 'log_prob' function. These proposal distributions can be accessed from the `glabcmcmc.distribution` module in the Python package. 'importance_proposal_base' needs to be chosen from `normflows.distributions.base` module.

3.2. Output

The generated Markov chain data will be returned and stored in 'output_dir/output_file', a CSV-formatted file, to facilitate subsequent processing and analysis of the results. Additionally, the posterior mean, variance, 95% credible interval, and effective sample size for each parameter will be displayed.

4. Example

Here we provide an example of utilizing functions of the `glabcmcmc` package to perform ABC inference. Specifically, we use a mixture model with four Gaussian peaks. The following codes illustrate the implementation of four different MCMC methods, the optimization of hyperparameters, and print the output of the GLMCMC approach. We also present the visualization of the GLMCMC results.

'Mixture_set()' is an ABCset interface that contains the key functions of the ABC model. Regarding ABCset, users can define it based on the format of 'Mixture_set()' according to target model.

```
from glabcmcmc import MCMCRunner
import glabcmcmc.distribution as distribution
from Mixture import Mixture_set
# Define ABC model set
```

```
Model = Mixture_set(epsilon=0.05)
theta0 = torch.tensor([0.0, 0.0])
# Set a random seed at the beginning to
# ensure the reproducibility of our results.
torch.manual_seed(0)
np.random.seed(0)
y0 = Model.generate_samples(theta0)
runner = MCMCRunner(Model, output_dir='./')
```

Next, we present a practical example of hyperparameter selection. In this example, all other hyperparameters of 'runner.run_glmcmc()' are constant, and optimization is performed solely on the global frequency parameter. The 'lp' refers to local proposal. The candidate parameter $\theta^* = \theta_{\text{old}} + z$, where θ_{old} is the current state of Markov chain and z is a random sample from local proposal 'lp'. The 'ip' refers to the importance proposal of iSIR. 'esjd()' computes the ESJD of a Markov chain. The selection of the optimal value for this parameter is guided by the rESJD (relative version of ESJD) criterion (Cao et al., 2024a) with fixed computational budgets.

```
from glabcmcmc import esjd
import numpy as np
import time
# Local proposal: theta_old + z, z~N((0,0),(0.35^2,0.35^2))
lp = distribution.DiagGaussian(2, loc=torch.zeros(1, 2),
        log_scale=torch.log(torch.tensor([0.35, 0.35])))
# Importance proposal of iSIR
ip = distribution.DiagGaussian(2, torch.tensor([0.0, 0.0]),
        torch.tensor([0.0, 0.0]))
seeds = np.linspace(1, 10, num=10)
batch_size = 5
global_frequencies = np.linspace(0, 1, num=11)
id = list(range(len(global_frequencies)))
resjd_value = [[0 for _ in id] for _ in range(len(seeds))]
num_ite2 = 1000
for i in range(len(seeds)):
    torch.manual_seed(seeds[i])
    for j in id:
        gf = global_frequencies[j]
        start_time = time.time()
        chain = runner.run_glmcmc(num_ite2, theta0, y0, gf,
                lp, ip, batch_size, output_file=None)
        end_time = time.time()
```

```
        time_mean =(end_time - start_time)/num_ite2
        resjd_value[i][j] = esjd(chain)/time_mean
resjd_mean = np.mean(resjd_value,axis=0)
best_gf = global_frequencies[np.argmax(resjd_mean)]
print(f"The best global frequency: {best_gf}")
```

The output is

```
The best global frequency: 0.9
```

For scenarios involving the optimization of multiple hyperparameters, a similar methodology can be employed, either through sequential tuning or joint optimization, to further enhance the model's overall performance.

Subsequently, the program is executed using the selected hyperparameter values.

```
chain_glmcmc = runner.run_glmcmc(1000000, theta0, y0, 0.9,
                                 lp, ip, 5)
```

Running this function will print information including the posterior mean, variance, and the 95% credible interval. Additional information can be obtained by directly manipulating the returned value, 'chain_glmcmc', or by processing the Markov chain data saved in the '.csv' file. When we run multiple MCMC chains, there might exist slight differences among different chains due to the intrinsic randomness. Here, we set a random seed prior to executing the MCMC chain to ensure the reproducibility of the results.

```
Theta_Re 1:
 Mean: -0.0017
 Variance: 2.0824
 95% Confidence Interval: (tensor(-2.8301), tensor(2.8268))
Theta_Re 2:
 Mean: -0.0259
 Variance: 2.0957
 95% Confidence Interval: (tensor(-2.8632), tensor(2.8115))
```

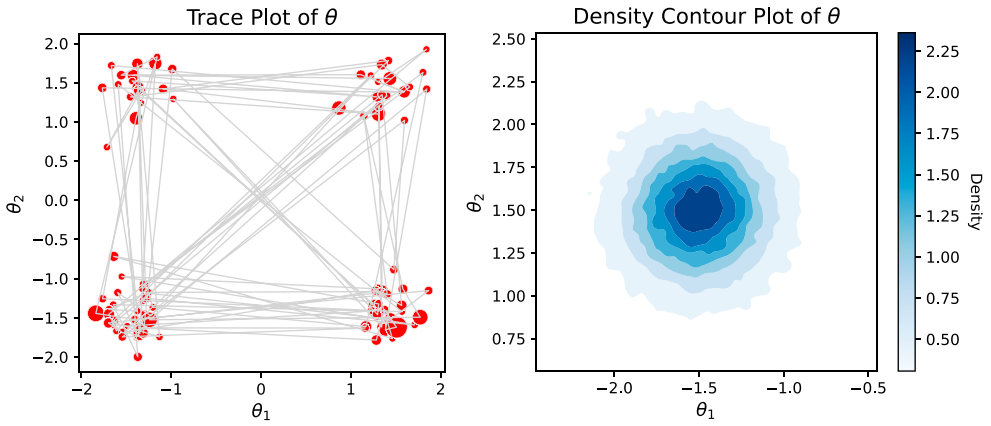

Figure 1. Trace plots (left) and posterior density contour (right) estimated by GLMCMC. The red dots and their size represent location and number of particles, and the grey lines depict the movement trajectories.

We visualize the posterior samples. Figure 1 displays part of the trace plots (left) (i.e., iteration 30001 $\sim$ 40000) and the posterior density contour (right) estimated by GLMCMC.

For other GL-ABC-MCMC methods, the following code provides an example. The outputs of these functions are similar to that of 'runner.run_glmcmc()'. In the method of updating the proposal distribution using normalizing flows, the initial proposal distribution should be selected from the `normflows.distribution.base` module.

```
gp = distribution.DiagGaussian(2, torch.tensor([0.0, 0.0]),
                                  torch.tensor([0.0, 0.0]))
chain_global = runner.run_global_mcmc(num_ite, theta0, y0,
                                       0.5, lp, gp)
chain_glmala =runner.run_glmala(num_ite, theta0, y0, 0.8,
                                   ip, 5, 0.3, 100)
import normflows as nf
gp_base = nf.distributions.base.DiagGaussian(2)
chain_glmcmc_nf =runner.run_glmcmc_nf(num_ite, 0.5, lp, 5,
                                       gp_base, 200, 50)
```

In addition, we infer the mixture of Gaussian posterior using 'ABC_mcmc' function of the R package `EasyABC` (Easy, 2013). This function implements the ordinary ABC-MCMC algorithm (Marjoram et al., 2003; Wegmann et al., 2009). We ensure that the length of the MCMC chain is consistent with our algorithm. The visualization of the posterior samples is shown in Figure 2. The results demonstrate that for multimodal posterior distributions, our algorithm exhibits superior inference performance. It effectively navigates the entire posterior space rather than becoming confined to a local region. Furthermore, our algorithm can also implement ordinary ABC-MCMC by setting the global frequency to zero.

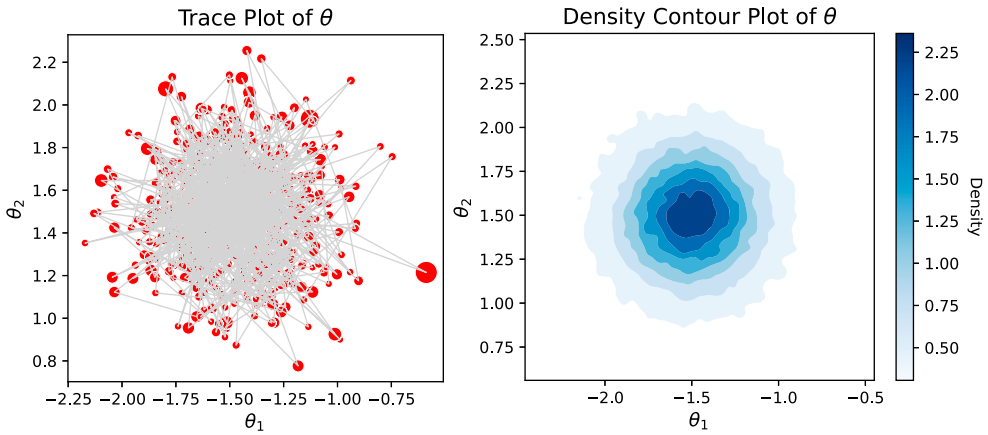

Figure 2. Trace plots (left) and posterior density contour (right) estimated by 'ABC_mcmc' from the R package `EasyABC`. The red dots and their size represent location and number of particles, and the grey lines depict the movement trajectories.

5. Conclusion

In this paper, we presented the `glabcmcmc` Python package, which implements a novel ABC-MCMC approach by integrating global and local proposal strategies. This innovative framework addresses the limitations commonly faced by traditional ABC-MCMC methods, particularly the challenges of slow convergence and entrapment in local regions. The package provides a comprehensive framework for the GL-ABC-MCMC algorithm, allowing users to flexibly call upon different sampling methods according to their specific goals and modelling needs.

Acknowledgments

The authorship is listed in alphabetic order.

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

This work was supported by the National Natural Science Foundation of China [grant numbers 12131001 and 12101333], the startup fund of ShanghaiTech University, the Fundamental Research Funds for the Central Universities, LPMC, and KLMDASR.

ORCID

Shijia Wang http://orcid.org/0000-0003-0339-1716

References

Barthelmé, S., & Chopin, N. (2014). Expectation propagation for likelihood-free inference. *Journal of the American Statistical Association*, *109*(505), 315–333. https://doi.org/10.1080/01621459.2013.864178

Beaumont, M. A., Zhang, W., & Balding, D. J. (2002). Approximate Bayesian computation in population genetics. *Genetics*, *162*(4), 2025–2035. https://doi.org/10.1093/genetics/162.4.2025

Cao, X., Wang, S., & Zhou, Y. (2024a). An adaptive approximate Bayesian computation MCMC with Global-Local proposals. *arXiv:2412.15644*.

Cao, X., Wang, S., & Zhou, Y. (2024b). Using early rejection Markov chain Monte Carlo and Gaussian processes to accelerate ABC methods. *Journal of Computational and Graphical Statistics Online*. https://doi.org/10.1080/10618600.2024.2379349

Csilléry, K., François, O., & Blum, M. G. (2012). ABC: An R package for approximate Bayesian computation (ABC). *Methods in Ecology and Evolution*, *3*(3), 475–479. https://doi.org/10.1111/mee3.2012.3.issue-3

Easy, A. B. C. (2013). Performing efficient approximate Bayesian computation sampling schemes using R. *Methods in Ecology and Evolution*, *4*(7), 684–687.

Liepe, J., Barnes, C., Cule, E., Erguler, K., Kirk, P., Toni, T., & Stumpf, M. P. (2010). ABC-SysBio–approximate Bayesian computation in Python with GPU support. *Bioinformatics (Oxford, England)*, *26*(14), 1797–1799.

Marjoram, P., Molitor, J., Plagnol, V., & Tavaré, S. (2003). Markov chain Monte Carlo without likelihoods. *Proceedings of the National Academy of Sciences*, *100*(26), 15324–15328. https://doi.org/10.1073/pnas.0306899100

Picchini, U. (2014). Inference for SDE models via Approximate Bayesian Computation. *Journal of Computational and Graphical Statistics*, *23*(4), 1080–1100. https://doi.org/10.1080/10618600.2013.866048

Pritchard, J. K., Seielstad, M. T., Perez-Lezaun, A., & Feldman, M. W. (1999). Population growth of human Y chromosomes: A study of Y chromosome microsatellites. *Molecular Biology and Evolution*, *16*(12), 1791–1798. https://doi.org/10.1093/oxfordjournals.molbev.a026091

Raynal, L., Marin, J. M.., Pudlo, P., Ribatet, M., Robert, C. P., & Estoup, A. (2019). ABC random forests for Bayesian parameter inference. *Bioinformatics (Oxford, England)*, *35*(10), 1720–1728.

Wegmann, D., Leuenberger, C., & Excoffier, L. (2009). Efficient Approximate Bayesian Computation coupled with Markov chain Monte Carlo without likelihood. *Genetics*, *182*(4), 1207–1218. https://doi.org/10.1534/genetics.109.102509 .https://doi.org/10.1534/genetics.109.102509

Wegmann, D., Leuenberger, C., Neuenschwander, S., & Excoffier, L. (2010). ABCtoolbox: A versatile toolkit for approximate Bayesian computations. *BMC Bioinformatics*, *11*, 1–7. https://doi.org/10.1186/1471-2105-11-116