MCMC4WF User Manual

Contents

1 Release notes **1**
1.1 Release 0.0.1 1

2 Installing Dependencies **1**

3 Installing `MCMC4WF` **2**

4 Calling `MCMC4WF` in `Python` **2**
4.1 The `MCMCOptions` class 3
4.2 The `MCMCSampler` class 4
4.2.1 `RunSampler` 4

5 Example `Python` scripts **5**

6 Bugs, queries, suggestions, comments **5**

1 Release notes

1. Initial release version 0.0.1, 17th February 2025

1.1 Release 0.0.1

This is the first version of `MCMC4WF` - below please find instructions on how to install the software on your machine, and how to use it within Python. For any enquiries or if you spot any bugs, please do not hesitate to contact me on Jaromir.Sant@gmail.com!

2 Installing Dependencies

`MCMC4WF` requires the following components to be run

1. `g++` compiler
2. `boost` library
3. `Python` together with `pip`/`pip3`
4. `CMake` together with `Ninja`
5. `pybind11`, `setuptools` and `wheel`
6. `EWF`

Please see the user manual for `EWF` [2] (`https://github.com/JaroSant/EWF`) for instructions how to install all of the above. Release 0.0.1 has been tested on the following versions of the above dependencies: `boost` 1.82, `Python` 3.12.3, `pip` 24.0, `CMake` 3.31.2, `Ninja` 1.11.1.3, `pybind11` 2.13.6, `setuptools` 75.6.0, `wheel` 0.45.1, `EWF` 1.3. Please note that new releases of the above software might require different installation procedures to the ones below, and thus there is no guarantee that these instructions are up to date nor correct for your specific platform!

3 Installing MCMC4WF

NB: if using Windows, please ensure that your PATH variable (part of your environment variables) is pointing at the directories containing g++, boost, python, pip, pybind11, CMake and EWF. You can find out where pybind11 and CMake are by running python3 -v and entering import PACKAGE_NAME.

The cleanest way to install MCMC4WF is through a Python virtual environment (venv). Note that this will install MCMC4WF *only within the virtual environment*, so you will need to activate said environment every time you wish to make use of MCMC4WF. To create a new venv:

1. Navigate to the root directory, for instance Documents/MCMC4WF.
2. Run python3 -m venv ENV_NAME where ENV_NAME is the desired name for the virtual environment
3. Activate the newly created venv by running source ENV_NAME/bin/activate

After doing this, you should be within the virtual environment and can run the command pip install --upgrade pip cmake ninja setuptools wheel pybind11 which ensures all necessary dependencies are installed into the virtual environment. Subsequently install EWF following the instruction manual provided for that software.

To install MCMC4WF, you can then run the following in terminal, assuming you are at the root location MCMC4WF/:

```
$ mkdir build
$ cd build
$ cmake ..
$ cmake --build .
$ cd ..
$ pip install .
```

Alternatively, you could simply run the above code snippet without creating the virtual environment, and which therefore installs the software globally on your machine.

Provided all steps are followed and no errors are thrown, then you can test that MCMC4WF was run correctly by running the provided test cases found in the examples directory, which should print some information to terminal, and create several texts files form which the relevant plots can be generated using the run_plotter.py script.

NB: To run the example script, you will need to have numpy and matplotlib installed in Python (simply run pip install PACKAGE in terminal).

4 Calling MCMC4WF in Python

To call MCMC4WF from within Python, simply add import MCMC4WF_pybind at the start of your Python script. This allows you to invoke the following classes: MCMCOptions and MCMCSampler. The former holds all the user-defined parameters for the MCMC algorithm (such as for instance prior means and standard deviations, proposal standard deviations, convergence precision, etc), whilst the latter possesses the runner function RunSampler which takes a dataset of your choice together with an appropriately set up MCMCOptions instance, and runs the algorithm.

For convenience, we recall that the Wright–Fisher diffusion with selection and mutation is given by the solution $X := (X_t)_{t \geq t_0}$ to the stochastic differential equation

$$dX_t = \frac{1}{2}\Big(\sigma X_t(1-X_t)\eta(X_t) - \theta_2 X_t + \theta_1(1-X_t)\Big)dt + \sqrt{X_t(1-X_t)}dW_t, \qquad X_0 = x,$$

where t_0 is the time at which the allele is born, $(W_t)_{t\geq 0}$ is a standard Brownian motion, σ is the selection coefficient, $\eta(x) := \sum_{i=0}^{d} \eta_i x^i$ for $d \in \mathbb{Z}_+$ constant, $\boldsymbol{\theta} := (\theta_1, \theta_2)$ is the vector of mutation parameters, and $x \in [0, 1]$ is a given starting point. We further specify that in our setup we assume that $X_s \equiv 0$ for any $s < t_0$.

4.1 The `MCMCOptions` class

Initialise a `MCMCOptions` class through which user-defined parameters are specified for use within the MCMC algorithm. It is important that all parameters are specified and set to some non-empty value, regardless of whether they are actually used or not (for instance even if one is interested in performing inference solely on σ, the relevant quantities on h and $\boldsymbol{\eta}$ need to be passed!)

Parameters:

- `sOptions` (*array double*) - an array of seven doubles specifying all the relevant information regarding the selection coefficient σ, ordered as follows: prior mean, prior standard deviation, prior lower bound, prior upper bound, proposal standard deviation, precision for mean convergence, precision for standard deviation convergence. **NB**: Whether the prior mean and standard deviation or the lower and upper bounds on the prior are used is decided by the `selTypePriors` variable (see below for full details), however all seven quantities need to be provided (even if they remain unused) for the algorithm to work! The same applies for the below `hOptions` and `etaOptions`
- `hOptions` (*array double*) - an array of seven doubles specifying all the relevant information regarding the dominance parameter h, ordered as follows: prior mean, prior standard deviation, prior lower bound, prior upper bound, proposal standard deviation, precision for mean convergence, precision for standard deviation convergence.
- `etaOptions` (*array of array doubles*) - an array of seven arrays containing doubles. Each array specifies all the relevant information regarding the coefficients of the selection function η, ordered as follows: prior mean, prior standard deviation, prior lower bound, prior upper bound, proposal standard deviation, precision for mean convergence, precision for standard deviation convergence. Within each array, the coefficients are ordered in increasing order of power, so the first entry is η_0, the second entry is η_1, and so on.
- `t0Options` (*array double*) - an array of seven doubles specifying all the relevant information regarding the allele age t_0, ordered as follows: prior mean, prior standard deviation, proposal standard deviation, precision for mean convergence, precision for standard deviation convergence.
- `theta` (*array double*) - an array of size 2 specifying the user pre-defined mutation parameter $\boldsymbol{\theta}$.
- `AlleleAgeMargin` (*double*) - a double (default setting is 1e-4) to avoid numerical instabilities.
- `AlleleAgePrior` (*int*) - an integer which specifies whether we assume Exponential (`AlleleAgePrior` = 0) or Gamma (`AlleleAgePrior` = 1) priors for the allele age t_0. If set to 0, then the first entry in `t0Options` is used as the parameters for the corresponding exponential distribution, whereas if it is set to 1, then the first and second entries of `t0Options` define the shape and scale parameters of the corresponding gamma distribution.
- `burnIn` (*int*) - integer specifies the amount of samples we throw away as burn-in.
- `lookBack` (*int*) - an integer specifying the window length along which the convergence criteria are computed.
- `printCounter` (*int*) - an integer specifying how often output is printed to terminal.
- `save` (*bool*) - a boolean specifying whether the user desires to save output

- `saveAux` (*bool*) - a boolean specifying whether the user wishes to save output from the auxiliary variables (namely the skeleton points) to file, useful for inspecting performance of the algorithm beyond simple posterior plots.
- `saveLikelihood` (*bool*) - a boolean specifying whether the likelihood evaluations for both selection and allele age updates should be computed.
- `selTypePrior` (*string*) - a string specifying whether Gaussian or Uniform priors will be used for the selection coefficients $\sigma, h, \boldsymbol{\eta}$. Setting this variable to '`Gaussian`' means that the first two entries of `sOptions`, `hOptions` and `etaOptions` are used to obtain the prior mean and standard deviation, whereas setting it to '`Uniform`' leads to the third and fourth entries being used as lower and upper bound on the uniform priors.
- `diffusion_threshold` (*double, optional*) - threshold below which Gaussian approximations are used in diffusion simulations. If omitted, will default to 0.1.
- `bridge_threshold` (*double, optional*) - threshold below which bridge approximations are used in diffusion bridge simulations. If omitted will default to 0.05.

Returns: Sets the file save names based on the machine time when invoked and the selected quantities to save.

4.2 The `MCMCSampler` class

This is a very simple wrapper for the MCMC algorithm, taking as input the dataset on which the user wishes to run the method.

Parameters:

- `Data` (*array int*) - an array of integers specifying how many samples have the allele being tracked at each observation time.
- `Samples` (*array int*) - an array of integers specifying the sample sizes observed at each observation time.
- `Times` (*array double*) - an array of doubles specifying the observation times *in diffusion time units*.
- `selType` (*int*) - integer specifying the selection regime the user wishes to infer. For genic selection (i.e. $\sigma \neq 0$, $\eta(x) \equiv 1$), this should be set to 0, for diploid (i.e. $\sigma \neq 0$ and $\eta(x) = x + h(1-2x)$ it should be set to 1 whereas any higher order polynomial selection (i.e. $\sigma \neq 0$ and $\eta(x) = \sum_{i=0}^{d} \eta_i x^i$) is coded as 2.

Returns: An instance of the `MCMCSampler` class

4.2.1 `RunSampler`

Sole member function of the `MCMCSampler` class which runs the MCMC algorithm on the previously specified dataset and user provided options.

Parameters:

- `MCMCOptions` (*class*) - structure specifying all the relevant options necessary for the algorithm to run.

Returns: Resulting output form the MCMC simulation is saved to file in the relevant files and locations (whose exact name and location are printed to terminal).

5 Example `Python` scripts

In the examples directory we provide a basic `Python` script `MCMCHorse.py` detailing how to use `MCMC4WF` from within `Python` on the horse coat coloration dataset analysed in the manuscript. After setting the desired options for the MCMC run through the `MCMCOptions` class, we initialise an instance of an `MCMCSampler` class using the dataset reported in [1], and subsequently run the MCMC routine by invoking the member function `RunSampler`.

To run the example scripts, please ensure your `Python` has the `numpy` and `matplotlib` packages installed (run `pip install numpy matplotlib` in terminal with the virtual environment active).

6 Bugs, queries, suggestions, comments

If you spot any bugs, or have any queries, suggestions or comments please do not hesitate to get in touch on Jaromir.Sant@gmail.com!

References

[1] A. Ludwig, M. Pruvost, M. Reissmann, N. Benecke, G. A. Brockmann, P. Castaños, M. Cieslak, S. Lippold, L. Llorente, A.-S. Malaspinas, M. Slatkin, and M. Hofreiter. Coat color variation at the beginning of horse domestication. *Science*, 324(5926):485–485, 2009.

[2] J. Sant, P. A. Jenkins, J. Koskela, and D. Spanò. EWF: simulating exact paths of the Wright–Fisher diffusion. *Bioinformatics*, 39(1):btad017, 01 2023.