# Working with the config file in ClimBayes

## Working with the config file

The function `ebm_fit` depends on parameters than can be specified in a configuration file, which is given in a YAML-format. It can be read via `yaml::read_yaml` and gives a nested list with the necessary settings to run the model and Bayesian inference of parameters. One such file can contain several different configurations, e.g.,

- "default"-configuration: Default parameters which we found generally useful for analysis of temperature observations.
- "experimental": A similar configuration which we found optimal for fast testing experiments.

! Please note, that once the default config of the package is changed, you'd need to go back to the github release to restore the default configuration. We therefore suggest to leave the default configuration of the package unchanged and create new config files to run own experiments. Below, we explain the structure of the file and how to create it.

The configuration can be loaded in the `ebm_fit` function. Firstly, you need to pass the path to config file to the `config_file` argument as a string. The config file that is already included in ClimBayes can be found via `system.file('extdata/ebm_fit_config.yml', package = 'ClimBayes')`, but you could also create your own config files locally and specify a different path. Secondly, you can specify the name of the configuration with `config` (e.g. `config = "default"`). Alternatively, you can pass a nested list (output of the `yaml::read_yaml`) to `config_file` (see below).

! Please note that all numbers needs to be given as floats, e.g., for `1` use `1.0`. Moreover, `yaml` does not support scientific notation, e.g. for `1e3` enter `1000.0`.

### Specifying estimated parameters

You can choose which parameters of the multibox EBM should be fixed and which should be estimated. By default, the weights, feedback parameters and initial forcing is estimated, whereas the heat capacity is fixed. Change this by switching from "fixed" to "estimate" in the parameters part, as, for example, here:

```
# specification of additional EBM parameters
# can either be 'estimate' or 'fixed'
# if 'estimate', then the value is estimated via the Bayesian approach
# if 'fixed', then the default values are taken from parameter_defaults
parameters:
  # heat capacity
  Cap: estimate
  # initial temperature parameter
  T0: estimate
  # initial forcing parameter
  F0: estimate
  # weights between feedback parameters
  weights: estimate
```

This part of the config file now defines that the heat capacity will be also estimated.

**Provide information about prior distributions**

The prior distribution can be chosen between a uniform and a beta distributed prior. For the beta distribution (see, e.g., https://en.wikipedia.org/wiki/Beta_distribution), the shape parameters can be separately chosen for all parameters to be estimated. Below, we show the part of the config file that can be modified to specify the prior distribution.

```
# specification of prior distributions
priors:
  # type, one of 'uniform' or 'beta'
  type: uniform
  # if type = 'beta', specify shape here
  beta_shape1:
    lambda:
      one_box:
      - 2
      two_box:
      - 2
      - 2
    weights:
      two_box:
      - 2
      T0: 2
      F0: 2
      Cap: 2
  beta_shape2:
    lambda:
      one_box:
      - 2
      two_box:
      - 2
      - 2
    weights:
      two_box:
      - 2
      T0: 2
      F0: 2
      Cap: 2
  # bounds for prior intervals
  one_box:
    lambda1_bounds:
      - 0.005
      - 2.0
  two_box:
    lambda1_bounds:
      - 0.2
      - 2.0
    lambda2_bounds:
      - 0.005
      - 0.2
  three_box:
    lambda1_bounds:
      - 0.2
      - 2.0
```

```
    lambda2_bounds:
      - 0.02
      - 0.2
    lambda3_bounds:
      - 0.005
      - 0.02
  T0_bounds:
    - -1.0
    - 1.0
  F0_bounds:
    - -2.0
    - 2.0
  # prior bounds for heat capacity
  # only required if the above default value is changed to 'estimate'
  Cap_bounds:
    - 5.0
    - 15.0
```

**Options for noise**

Here, we explain the options for the choice of noise process used in the likelihood function. To this end, we provide a little more background on Bayesian inference (for details see our submitted manuscript) . Assume we infer the posterior of uncertain parameters $\theta$ conditioned on some observational data $Y = y$. Bayes theorem states that

$$p(\theta|y) \propto p(y|\theta)p(\theta).$$

The density $p(y|\theta)$ is called the likelihood. To calculate the likelihood, assume

$$Y = \Phi(\theta) + Z(\theta),$$

where $\Phi$ is the so-called forward operator and $Z$ is a stochastic noise process that is also allowed to depend on $\theta$. In our case, $\Phi$ corresponds to the forced response of the EBM $T_{1,F}$ (see tutorial vignette and main manuscript). $Z$ can be a sum of the EBM's internal climatic dynamics $T_{1,I}$ and measurement uncertainties in observations. For computational reasons, we do not compute $Z(\theta)$ dynamically for each sample of the chain. Instead, we implement several options to approximate it (see Appendix of the manuscript for details).

**Option 1: Approximation using white noise** In the most simple case, we choose white noise as the noise process corresponding to random uncertainties in the observation, such that both internal dynamics as well as internal dynamics are modeled jointly by one white noise process. It is characterized by its mean (zero) and the standard deviation. The standard deviation (SD) can be represented in two ways in our package:

**1a: Fixed SD** The noise is set to white noise with a pre-specified SD. Choose parameters in the config file as follows:

```
noise:
    # type of noise
    # can be 'ar1_iterative', 'ar1_fixed', 'white_fixed' or 'white_iterative'
    type: white_fixed

    # if type = 'white_fixed', specify standard deviation of white noise
    white:
      SD: 0.1
```

**1b: Iteratively** The noise is set to white noise with a pre-specified initial SD. The algorithm runs once. Then, the SD of the residual (observations - EBM fit) is calculated, the noise is set to this value and the algorithm is run again. Of course, this could be repeated many times, but we typically find that one iteration suffices. The initial SD and `type="white_iterative"` can be chosen in the config file accordingly:

```
# parameters to specify the noise in the likelihood distribution
noise:
  # type of noise
  # can be 'ar1_iterative', 'ar1_fixed', 'white_fixed' or 'white_iterative'
  type: white_iterative


  # if type = 'white_iterative', specify inital value for standard deviation
  # of white noise
  white_iterative:
    SD: 0.1
```

**Option 2: Approximate using an AR(1) process** For this option, we model the noise as the sum of two noise processes: one corresponding to the internal dynamics plus one for the measurement noise. The noise arising from internal dynamics is approximated by a (discrete) AR(1) process (or OU-process in a continuous setting), which goes back to K. Hasselmann, "Stochastic Climate Models" (1976). For a more detailed physical motivation of this noise process see also our manuscript. Performing the stochastic extension of the energy balance model leads to an AR(1) process whose parameters depend on the EBM parameters $\lambda_k$ and $w_k$. The measurement noise is approximated by a white noise process with mean zero and a fixed SD, and we assume it to be independent of the internal dynamics.

**2a: Fixed parameters** The noise is set to the sum of AR(1) noise with pre-specified SDs and parameters $\lambda_k$ and $w_2, ..., w_N$ and white noise with pre-specified SD. This can be done as follows:

```
# parameters to specify the noise in the likelihood distribution
noise:
  # type of noise
  # can be 'ar1_iterative', 'ar1_fixed', 'white_fixed' or 'white_iterative'
  type: ar1_fixed

  # if type = 'ar1_fixed', specify standard deviation of white noise,
  # standard deviation of ar1 noise and correlation parameters
  ar1_fixed:
    SD_white: 0.01
    SD_ar1: 0.1
    lambda:
      one_box:
      - 0.3
      two_box:
      - 1.0
      - 0.05
    weights:
      two_box:
      - 0.8
```

**2b: Iteratively** Similar to option 1b, the noise is set to the sum of AR(1) noise with pre-specified initial SD, $\lambda_k$, and $w_k$ values and white noise with pre-specified SD. The algorithm is run once. Then, the SD of the residual (observations - EBM fit) is calculated. Together with the parameter estimates of $\lambda_k$ and $w_k$,

they are taken as input values for the next iteration. Here, the SD of the white noise is fixed, only that for the AR(1) noise is adjusted. The algorithm is run again. This iterative procedure can be repeated many times. Typically, one iteration suffices here, too.

The config file entries in this case look as follows (the initial values for $\lambda_k$ and $w_k$ are determined automatically):

```
# parameters to specify the noise in the likelihood distribution
noise:
  # type of noise
  # can be 'ar1_iterative', 'ar1_fixed', 'white_fixed' or 'white_iterative'
  type: ar1_iterative

  # if type = 'ar1_iterative', specify standard deviation of white noise (fixed),
  # and initial value of standard deviation of ar1 noise
  ar1_iterative:
    SD_white: 0.01
    SD_ar1_default: 0.1
```

### Options for MH-algorithm

In the `metropolis_hastings` (MH) part of the config file, you can specify parameters related to the MH algorithm (https://en.wikipedia.org/wiki/Metropolis%E2%80%93Hastings_algorithm, see also our manuscript), for example: - `n_chains`: the number of simulated chains - `burn_in_proportion`: the burn-in period (it is chosen to be `n_samples` / `burn_in_period`) - `n_samples`: the number of samples - `proposal_variance`: variances for normal proposal distribution - `alpha`: weighting factor in proposal distribution (see below) - `chain_start_buffer`: determines a buffer for the starting points of the chains, given as a proportion of the prior interval (for multiple chains, the starting points are chosen well-distributed over the entire prior interval, but sometimes moving the points closer to the middle can improve sampling behavior) - `dynamic_termination`: Boolean - determines if sampling should be continued until the termination criteria are met; if yes, `n_chains` needs to be greater than one, additional parameters are relevant: - `n_samples_dynamic`: number of samples to be calculated each time before termination criteria are checked again - `error_tolerance`: adjusts the width of a confidence interval for the posterior mean (see Flegal . . . ); the sampling is continued until the width of the confidence interval is about `error_tolerance * parameter values` - `gelman_rubin_diagnostic`: threshold for metric by Gelman and Rubin that assesses mixing behavior of chains; the sampling is continued until the metric is smaller than the specified threshold

A note on the proposal distribution: At the start of the algorithm, the proposal for the $j$-th sample is normally distributed around $x_{j-1}$. The covariance matrix is determined by $\mathrm{diag}(\boldsymbol{\sigma})$, that is the diagonal matrix with entries given by $\boldsymbol{\sigma}$ = `proposal_variance`. To summarize:

$$x'_j \sim \mathcal{N}(x_{j-1}, \mathrm{diag}(\boldsymbol{\sigma}))$$

To improve the convergence properties of the sampling scheme, after the first $m$ samples ($m$ is currently set to half of the burn-in period), we change the proposal distribution to

$$\mathcal{N}(x_{j-1}, \alpha \mathrm{Cov}(\mathrm{x_m}, ..., \mathrm{x_{j-1}}) + (1-\alpha)\mathrm{diag}(\boldsymbol{\sigma})),$$

for $j - 1 >> m$.

### Additional parameters

- The parameter `parallel` specifies if computation of multiple chains should be run in parallel. Via `parallel_free_cores` you can adjust how many cores should still be free, the computation will run on `parallel::detectCores() - parallel_free_cores` cores. Please note that this will use additional resources on your machine, yet it runs faster :)

- Via `return_solution_matrix`, you can choose if the result of `ebm_fit$samples` should contain the entry `model_fit` with the EBM solutions for all samples from the MCMC algorithm. This allows for more detailed analysis but potentially needs a lot of storage in case of long time series or many samples.

### Passing a nested list to `config_file` argument

As mentioned above, you can pass a nested list to `config_file`, as it is generated by `yaml::read_yaml` from the config file. This can be convenient if you wish to run many similar runs while modifying the config Instead of creating many versions of very similar configuration files, you could create one, read it in and modify only the list entries.

For example, let's study the influence of the heat capacity only:

```
# read in one config list
config_list <- yaml::read_yaml(system.file('extdata/ebm_fit_config.yml',
                                           package = 'ClimBayes'))
# change a parameter in the "experimental" config directly
config_list$experimental$parameter_defaults$Cap <- 9
# run the ebm_fit with this config list
ebm_fit(obs, forc, 1, config_file = config_list, config = "experimental")

# modify parameters and repeat
config_list$experimental$parameter_defaults$Cap <- 10
ebm_fit(obs, forc, 1, config_file = config_list, config = "experimental")

config_list$experimental$parameter_defaults$Cap <- 11
ebm_fit(obs, forc, 1, config_file = config_list, config = "experimental")
```