Intro to ClimBayes

This vignette

This vignette is a tutorial for the `ClimBayes` package. It provides explanations on how Bayesian inference for multibox energy balance models (EBMs) is implemented and how it can be used to study temperature time series. Please refer to the `README` of this package as well as the manuscript submitted to Chaos for all details.

Multibox EBMs

Consider the most simple zero-dimensional climate model for the global annual mean surface temperature T

$$\frac{dT}{dt}(t) = -\lambda T(t) + \frac{1}{C}F(t),$$

where λ is a climatic feedback parameter (the inverse of the characteristic timescale) and F is the external forcing.

It has the well-known solution

$$T(t) = \frac{1}{C}\int_{-\infty}^{t} R(t-s)F(s)ds$$

with the response function

$$R(t) = e^{-t\lambda}.$$

This model can be generalized to a multibox model by adding additional layers, characterized by their heat capacities. Typically, these layers are introduced to model the influence of the deep ocean (with a larger heat capacity) on the surface temperature. Neglecting internal variations and measurement errors, the surface temperature of a generic N-box model has the solution

$$T_{1,F}(t) = \frac{1}{C_1}\int_{-\infty}^{t} R(t-s)F(s)ds,$$

with response function

$$R(t) = \sum_{k=1}^{N} w_k e^{-t\lambda_k}.$$

C_1 is the heat capacity of the surface layer and λ_k gives N feedback parameters (their inverses are N characteristic timescales), weighted with w_k and $w_1 + ... + w_N = 1$.

The `ClimBayes` package estimates the parameters λ_k and w_k as well as an initial temperature T_0 and an initial forcing parameter F_0 by fitting multibox models to temperature data. Note that in the special case $N = 1$, there is only λ_1 and $w_1 = 1$, hence w_1 does not need to be estimated. If $N > 1$, it suffices to estimate $w_1, ..., w_{N-1}$ and then $w_N = 1 - (w_1 + ... + w_{N-1})$.

Bayesian inference

Bayesian inference aims to infer the posterior distribution of uncertain parameters conditioned on observational data. Here, it is used to compute the marginal posteriors of the EBM parameters and, hence, the best model fit. To this end, the implemented Bayesian approach combines the energy balance model, observational data and prior information on parameters via Bayes theorem. Computationally, a Metropolis-Hastings algorithm is used to find the optimal parameters. It is important to note, that "observational data" includes both the external forcing $F(t)$ as well as the observed temperature $T(t)$ and assumes a yearly resolution.

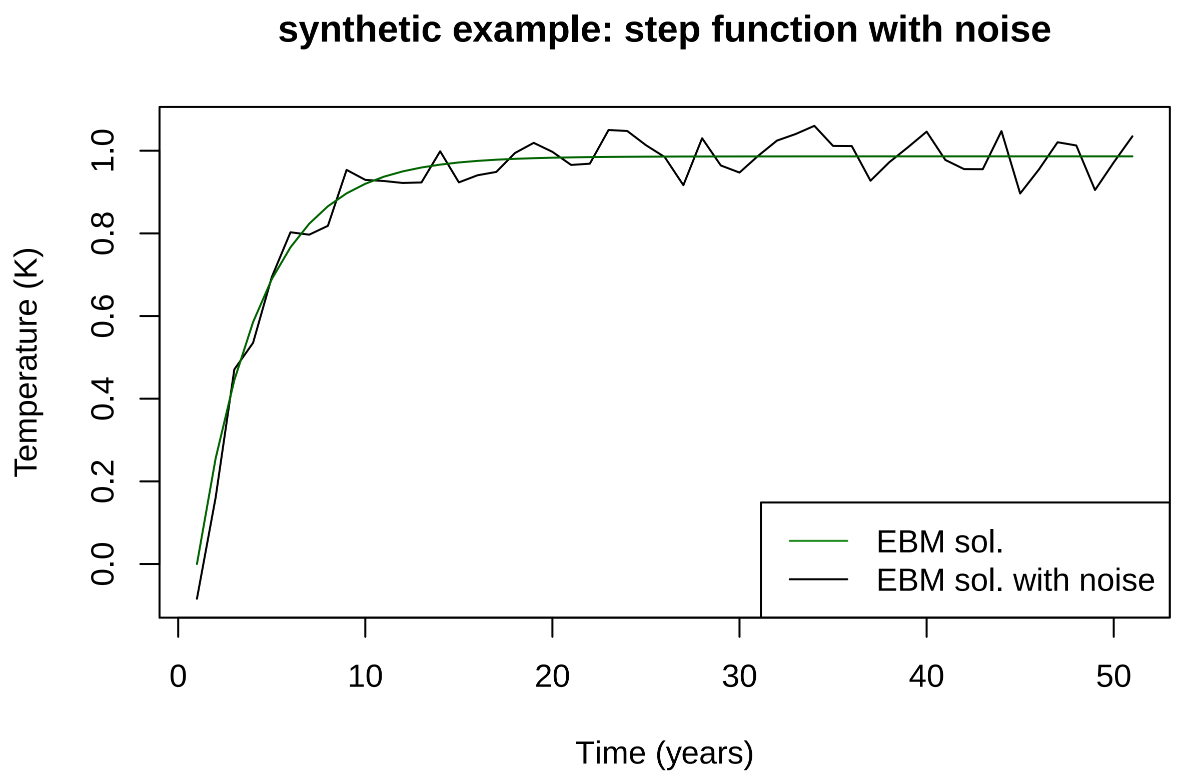
To summarize, the `ClimBayes` package can be used to estimate the climate parameters of an $N-$box model given the input $F(t)$ and output $T(t)$ as well as prior information on the parameters. First, we show how the model can be used to generate synthetic data given a set of parameters and arbitrary external forcing. Second, we explain how to perform this estimate using the HadCRUT5 data set and PMIP3 forcing reconstructions. The `config_file`-vignette additionally explains how own data sets can be included and documented in `*config.yml`.

Example: Synthetic data

We fix parameters and generate synthetic data by solving the EBM for a step function forcing with additional white noise.

```r
forcing <- c(0, rep(3, 50))
lambda <- 0.3
sd_white <- 0.05
# calculate solution for 1-bx model with given parameters
sol <- solve_ebm(lambda, weights = 1, T0 = 0, F0 = 0, Cap = 10.1,
                 forc_vals = forcing)
noise <- rnorm(51, 0, sd_white)

obs <- sol + noise
plot(obs, type = "l", xlab="Time (years)", ylab="Temperature (K)",
     main="synthetic example: step function with noise")
lines(sol, col = "darkgreen")
legend(x = "bottomright",          # Position
       legend = c("EBM sol.", "EBM sol. with noise"),
       col = c("forestgreen", "black"),
       lwd = 1)
```



Next, we use the `ebm_fit()`-function to estimate the parameters of the generated data set. As an input, the `ebm_fit()`-function takes the observational data (i.e. synthetically generated time series) and the step-function forcing. Moreover, the number of boxes `n_boxes` can be specified. The argument `config_file` specifies the path to the config file and `config = "experimental"` determines the config parameters we've chosen for quick and easy tries. Note that both `config_file = system.file('extdata/ebm_fit_config.yml', package = 'ClimBayes')` and `config = "experimental"` are also the default settings. Config files are explained in more detail in the `config`-vignette.

```
fit <- ebm_fit(obs, forcing,
               n_boxes = 1,
               detrending = FALSE,
               config_file = system.file('extdata/ebm_fit_config.yml', package = 'ClimBayes'),
               config = "experimental")
```

```
## Starting first iteration Starting second iteration
```

The function `ebm_fit()` returns an object of class `ebm_fit`, called `fit` here. The `fit` can be easily be printed

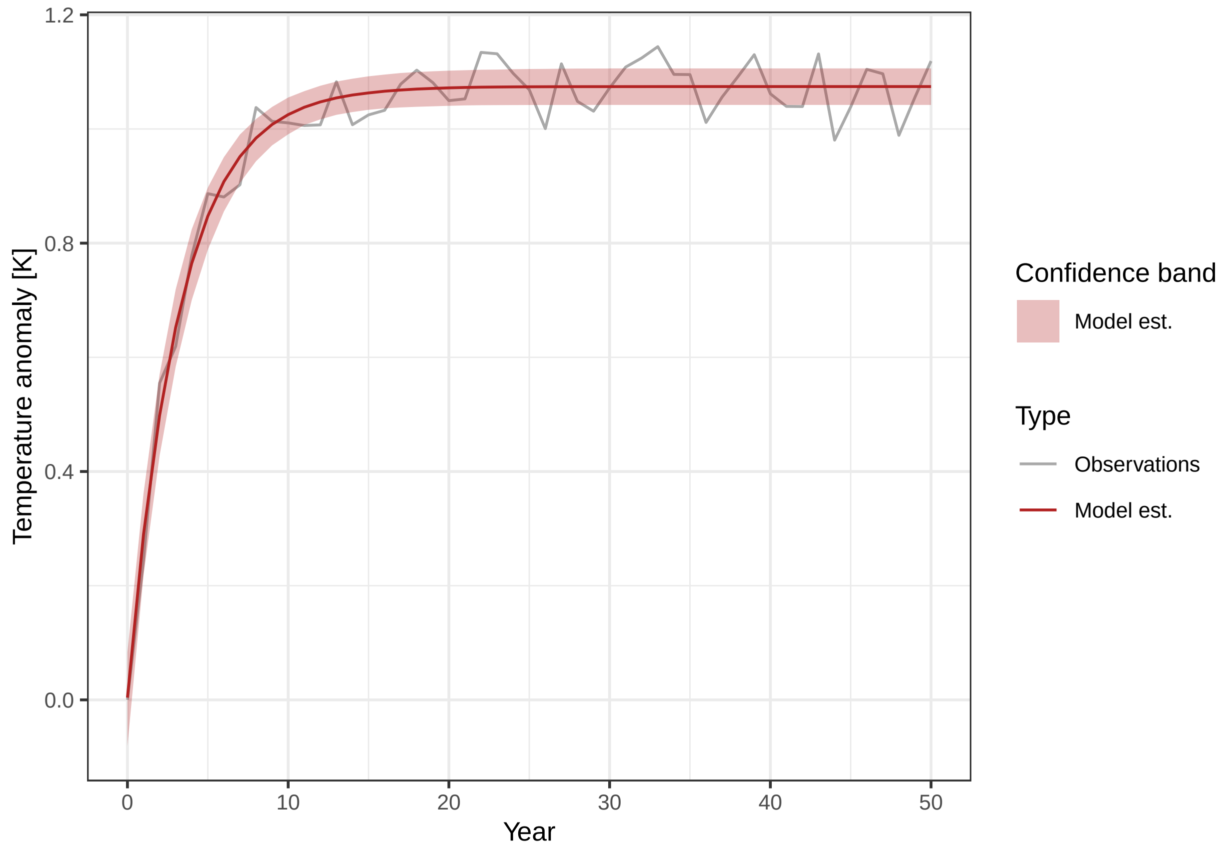
```
print(fit)
```

```
## Fitted 1-box EBM to data
## from years 0 to 50
## Estimated parameters:
## Parameter lambda1 with mean: 0.31
## Parameter T0 with mean: -0.038
## Parameter F0 with mean: 0.508
## RMSE: 0.041
```

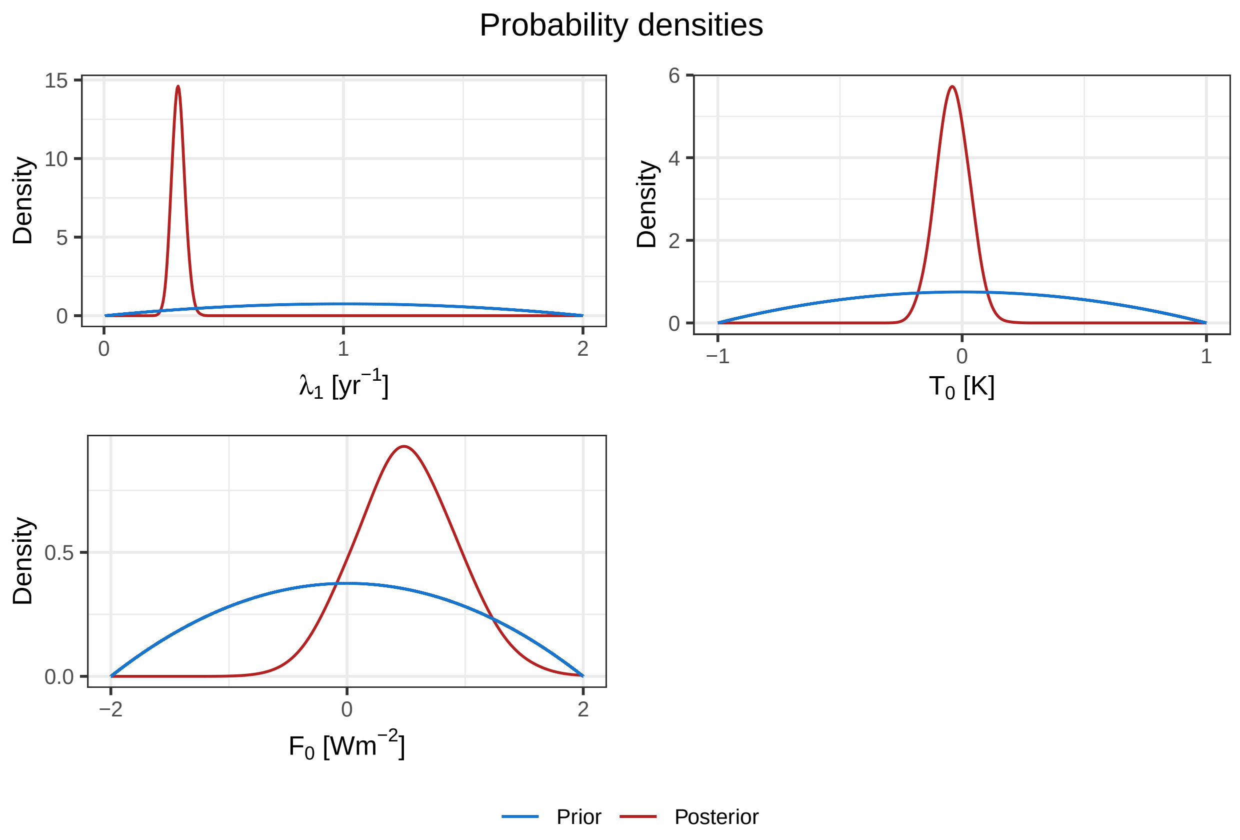
or plotted

```r
plot(fit)
```

```
## $gg_fit
```



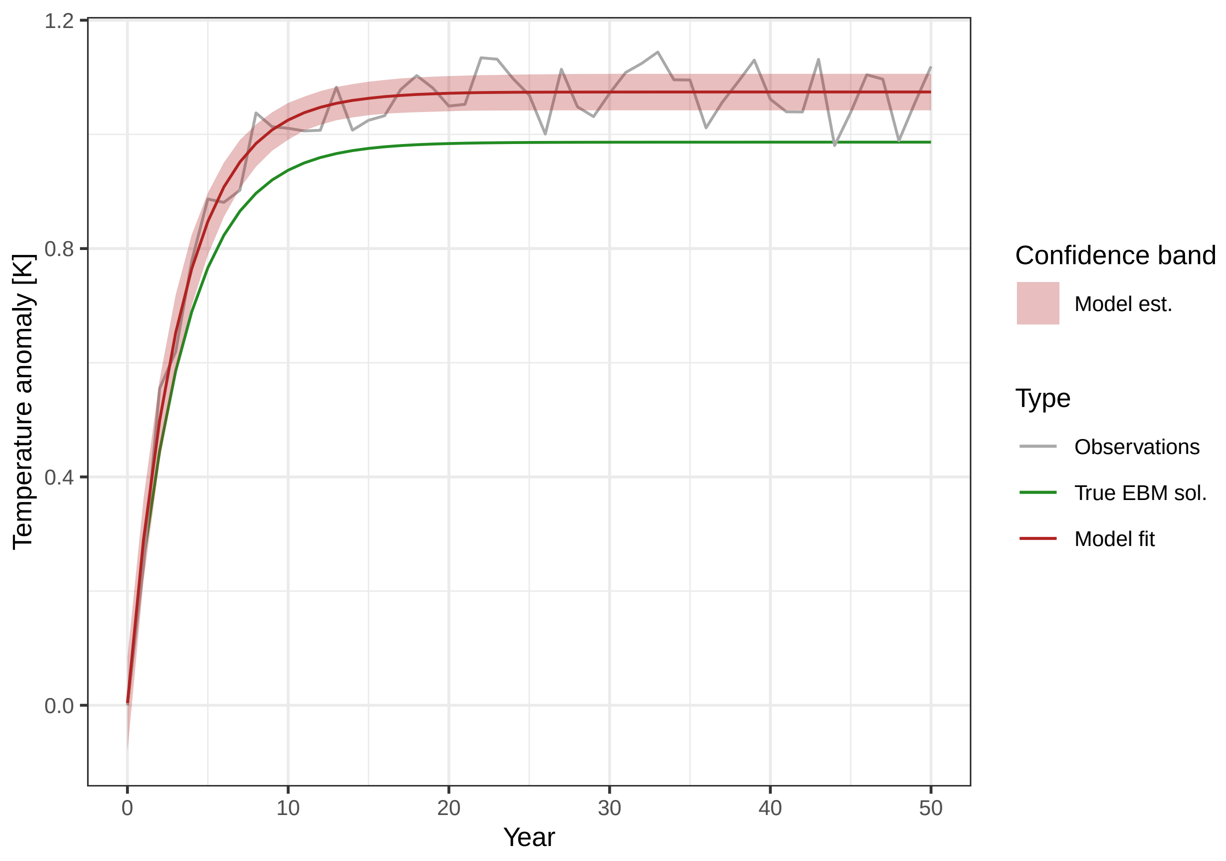
```
##
## $gg_marginals
```



Here, the `plot` function calls `plot_marginals`, which plots marginal posterior distributions of all estimated parameters, and `plot_fit`, which plots observations and the model fit including a 95%-credible interval.

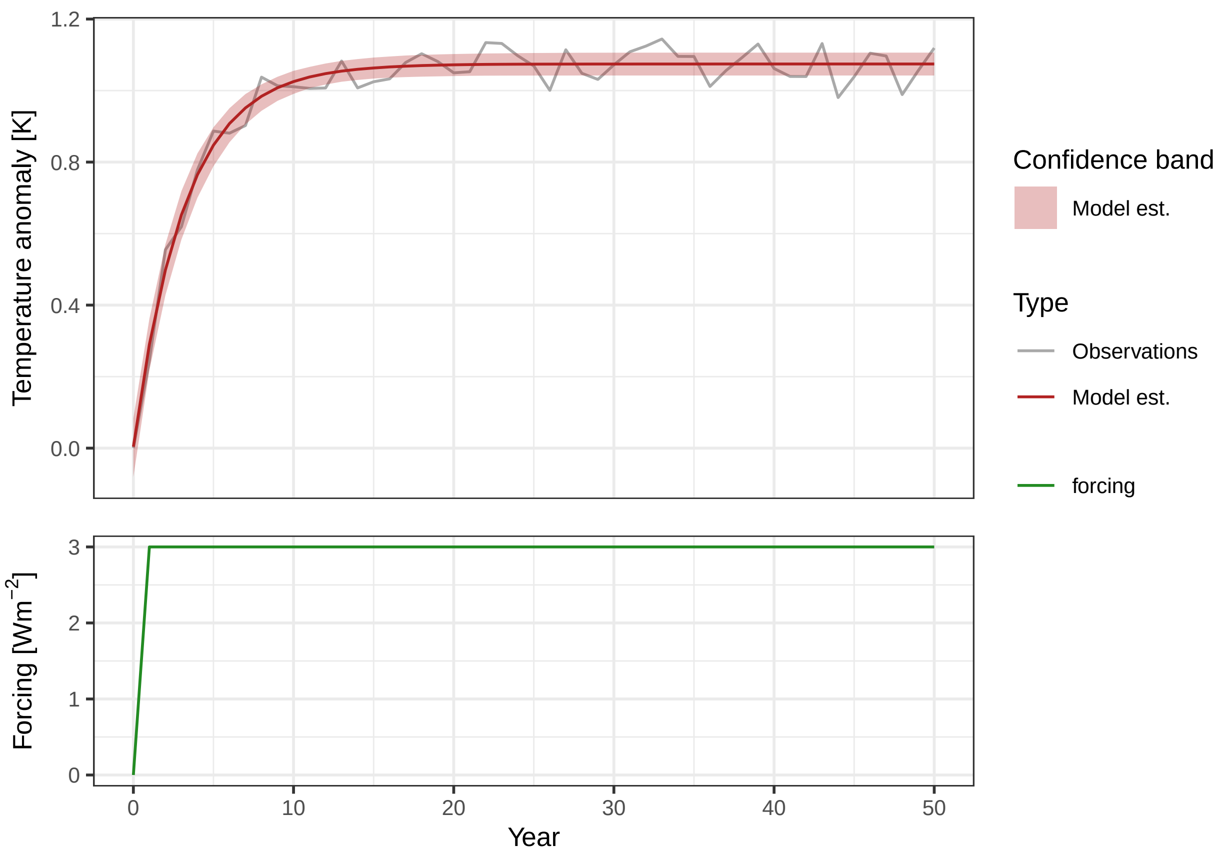
The functions `plot` and `plot_fit` have an additional argument `ebm_sol`, which takes another vector of temperature values that is then plotted alongside the observations and the model fit. If the synthetic observation data was generated as an EBM solution plus noise, you could set `ebm_sol` to the EBM solution without the noise.

```
plot_fit(fit, ebm_sol = sol)
```



Moreover, the forcing time series can be easily included into the plot:

```
plot_fit(fit, plot_forcing=T)
```



From the above plotted marginals as well as the print-out, we observe that the estimated parameter $\lambda 1$ is close to the data-generating parameter $\lambda = 0.3$. It is not exactly the same as we added white noise to the data before. The plot method shows the fit as well as marginal posteriors for all estimated parameters. The class objects are nested lists and contain additional information. For example, `fit$posteriors$parameters` gives the posterior medians, means and variances of all estimated parameters.

```
print(fit$posteriors$parameters$lambda1)
```

```
## $median
## [1] 0.3099703
##
## $mean
## [1] 0.3104738
##
## $variance
## [1] 0.0006406757
##
## $lower_quant
## [1] 0.2642467
##
## $upper_quant
## [1] 0.3627624
```

Example with historical data

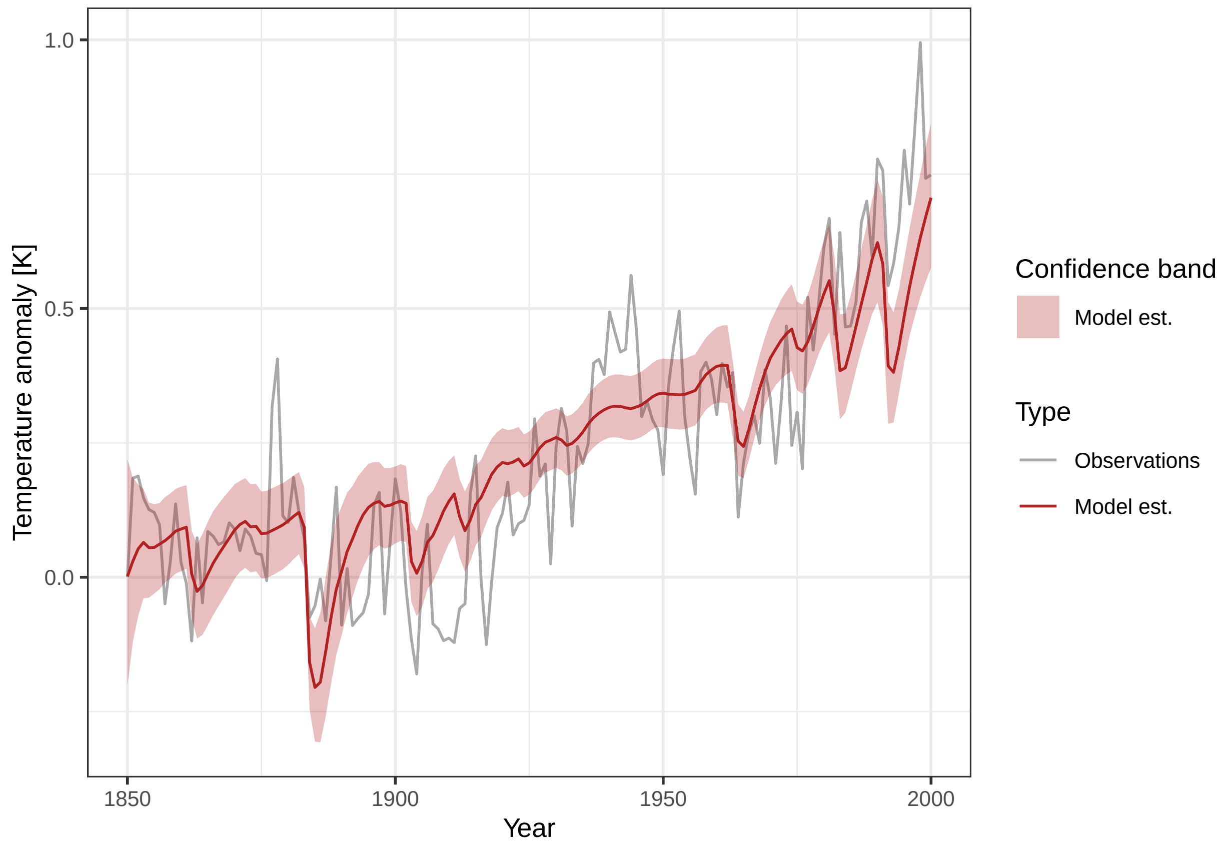
As a real-world example, we load HadCRUT5 observational data and the forcing reconstructions from Schmidt et al. Please find all references in the `README`. First, we load the data and then call the `ebm_fit` function. Most conveniently, observations and forcings should be provided as data frames with columns `year` and `temperature` / `forcing`. Additionally, the first and last year of the considered period for the fit can be adjusted.

```r
data("hadcrut")
data("schmidt")
fit_1box <- ebm_fit(hadcrut, schmidt,
             n_boxes = 1,
             start_year = 1850,
             end_year = 2000)
```

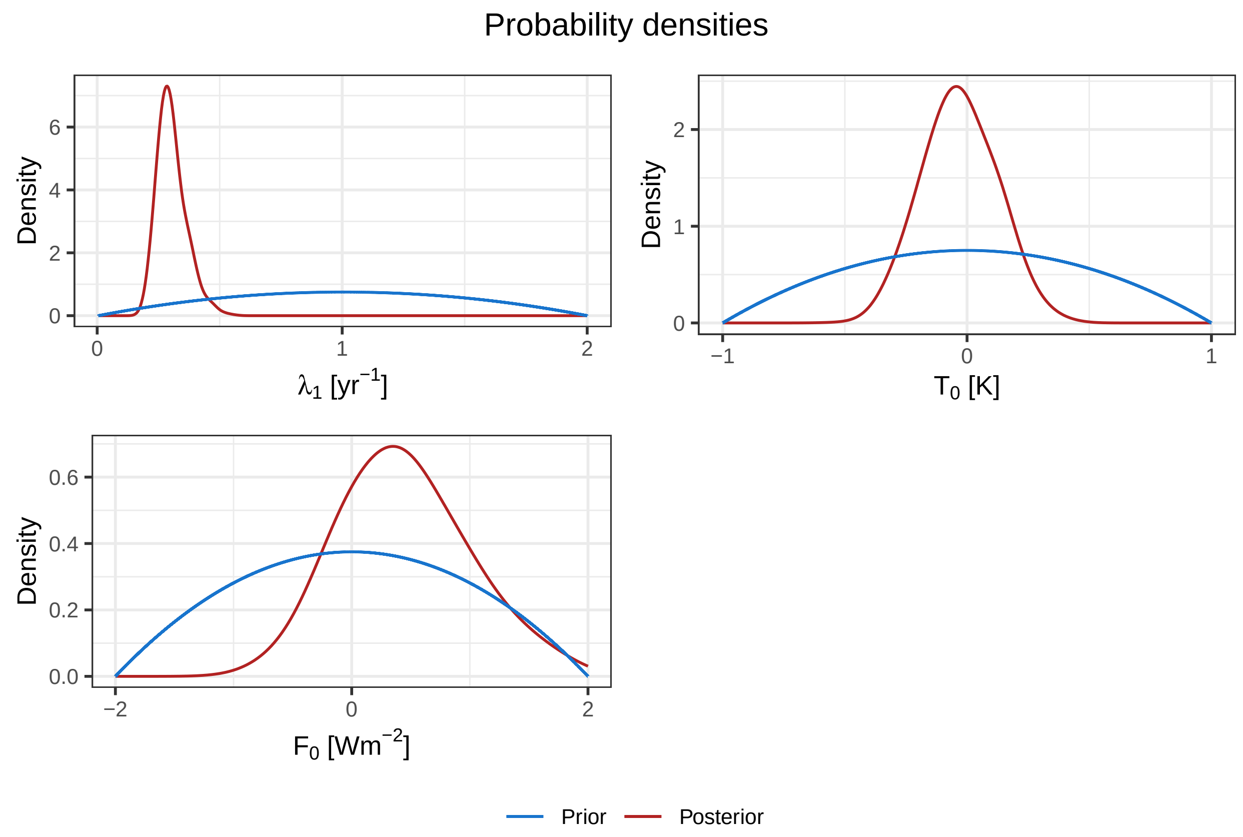
```
## Starting first iteration Starting second iteration
```

```r
plot(fit_1box)
```

```
## $gg_fit
```



```
##
## $gg_marginals
```



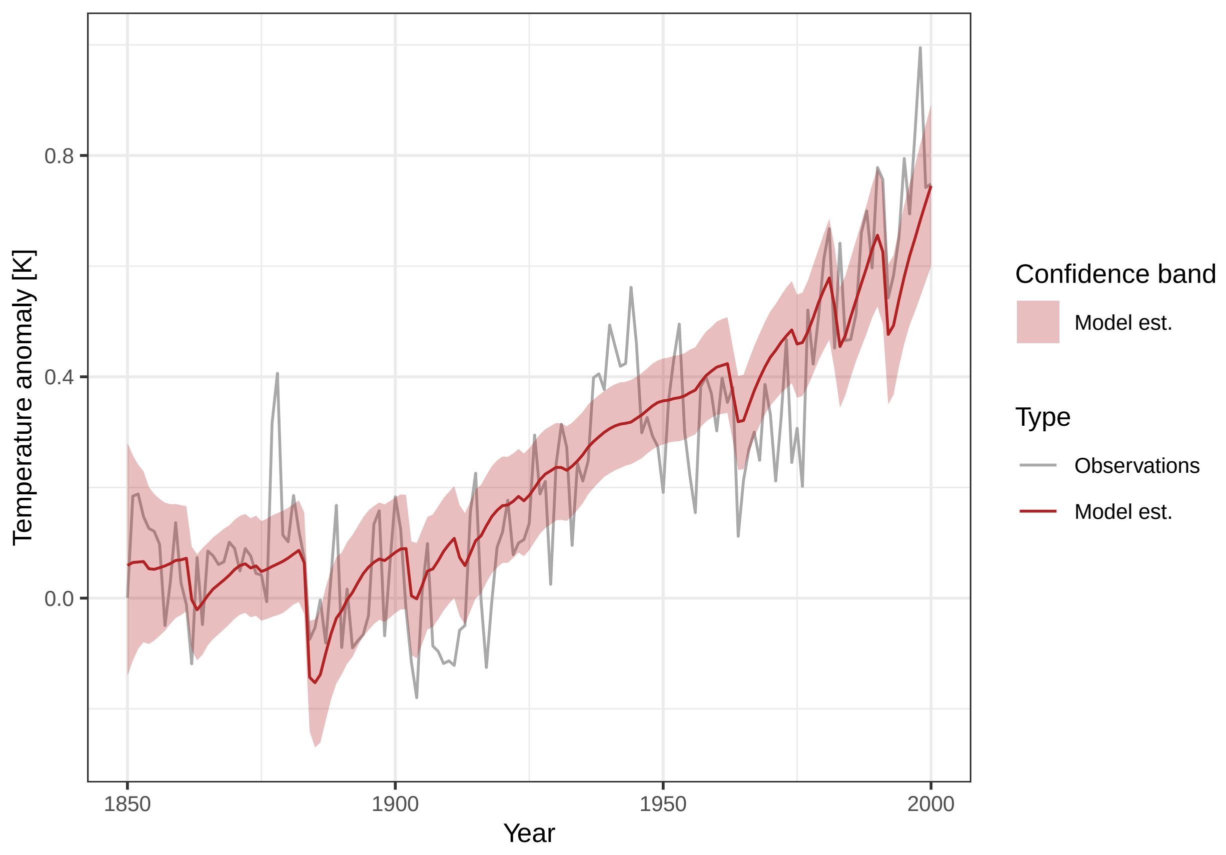
To compare the one and two box model, we change the parameter `n_boxes` and visualize the differences using `plot_two_fits()`.

```
data("hadcrut")
data("schmidt")
fit_2box <- ebm_fit(hadcrut, schmidt,
             n_boxes = 2,
             start_year = 1850,
             end_year = 2000)
```

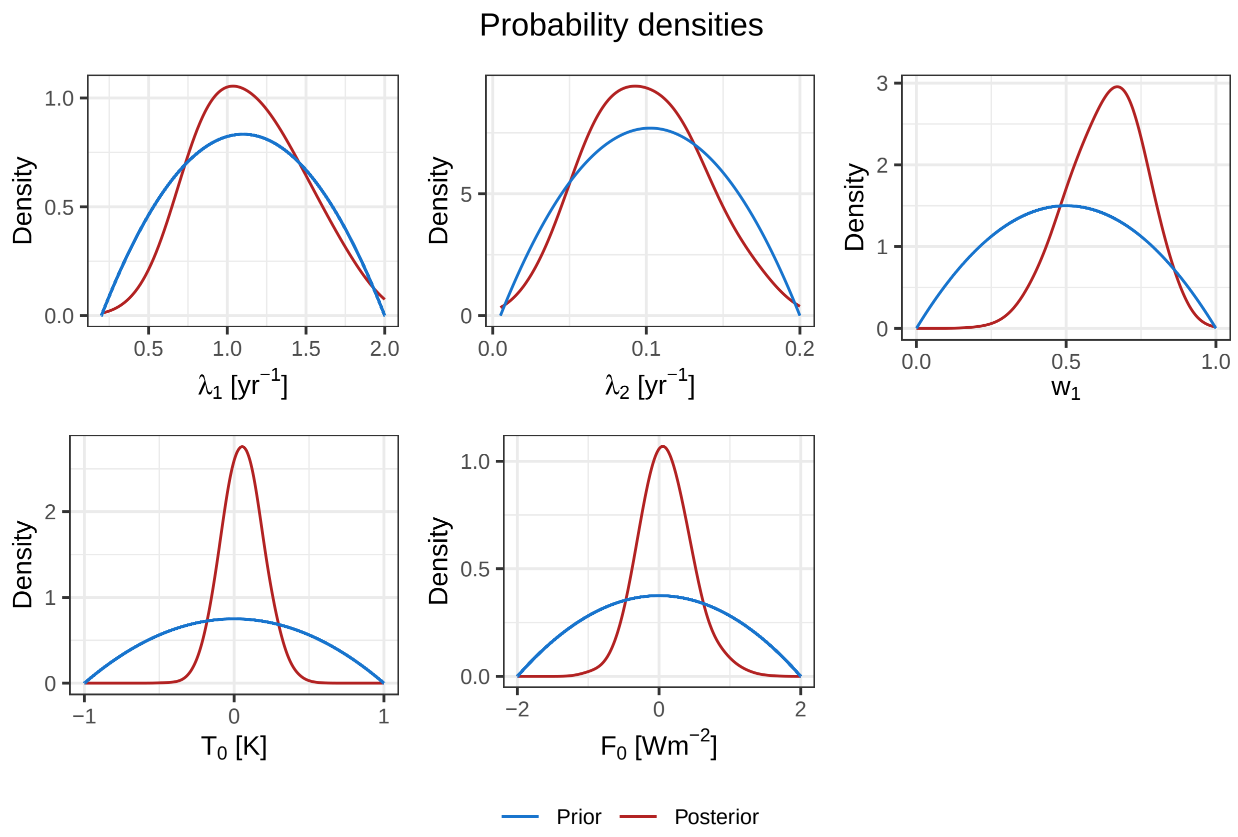
```
## Starting first iteration Starting second iteration
```

```
plot(fit_2box)
```

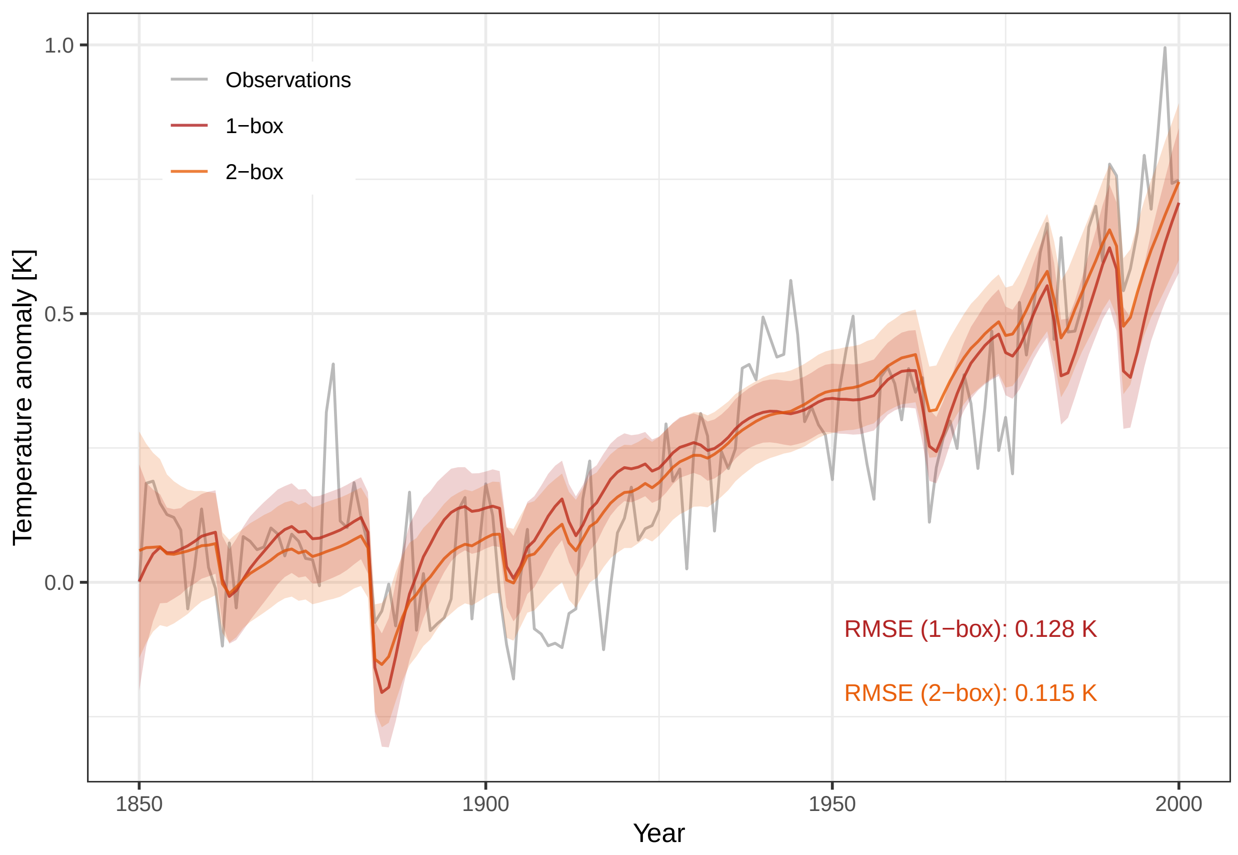
```
## $gg_fit
```



```
##
## $gg_marginals
```



```r
plot_two_fits(fit_1box, fit_2box)
```



Including own data sets

The `ebm_fit` function can work with vectors or data frames as inputs. Therefore, if you want to include your own data set, you have two options:

Option 1 (recommended): Save your data as a data frame with a column `year` and a column `temperature` (`forcing`) for the temperature (forcing) data. As an example, we store the temperature and forcing values from the above data sets for the years 1900-200 in the `temp_vals` and `forc_vals` vectors:

Next, we transform the vector to a dataframe and write it to `../data/`.

```
temperature_data <- data.frame(year = 1900:2000, temperature = temp_vals)
save(temperature_data, file = "../data/temperature_data.rda")
forcing_data <- data.frame(year = 1900:2000, forcing = forc_vals)
save(forcing_data, file = "../data/forcing_data.rda")
```

Using the same loading function from above and calling `ebm_fit` will fit the EBM to the entire time series without any additional arguments.

```
load("../data/temperature_data.rda")
load("../data/forcing_data.rda")
ebm_fit(temperature_data, forcing_data, 1)
```

```
## In ebm_fit: No years  for 'obs' supplied. Use entire time series.

## No years for 'forc' supplied. Use entire time series.

## Starting first iteration Starting second iteration
## Fitted 1-box EBM to data
## from years 0 to 100
## Estimated parameters:
## Parameter lambda1 with mean: 0.247
## Parameter T0 with mean: 0.005
## Parameter F0 with mean: -0.222
## RMSE: 0.121
```

As above, the span of the fit can be specified.

```
ebm_fit(temperature_data, forcing_data, 1,
        start_year = 1950,
        end_year = 1970)
```

```
## Starting first iteration Starting second iteration
```

```
## Fitted 1-box EBM to data
## from years 1950 to 1970
## Estimated parameters:
## Parameter lambda1 with mean: 0.905
## Parameter T0 with mean: 0.071
## Parameter F0 with mean: 0.348
## RMSE: 0.085
```

- Option 2: Run the function directly on the vectors of temperature and forcing data. However, specifying the start and end year won't be possible anymore. Nevertheless, this option might be more convenient when trying our synthetic data sets.

```
ebm_fit(temp_vals, forc_vals, 1)
```

Sampling of internal variability

As described in the main manuscript, the above multibox EBM can be extended to a stochastic model by adding random white noise forcing that represents weather fluctuations. Then, the surface temperature response is typically partitioned into internal and externally-forced components:

$$T_1(t) = T_{1,F}(t) + T_{1,I}(t) = \int_{-\infty}^{t} R(t-s)\frac{1}{C_1}F(s)ds + \int_{-\infty}^{t} R(t-s)\frac{\sigma_W}{C_1}dW(s)$$

The internal fluctuations $T_{1,I}(t)$ are described by an Itô-integral over the Wiener process $W(s)$ and σ_W gives the standard deviation of a white noise process (see main manuscript for details). ClimBayes allows for sampling from these internal variations using `ClimBayes::gen_noise_from_ebm_fit()`), as shown in the below example.

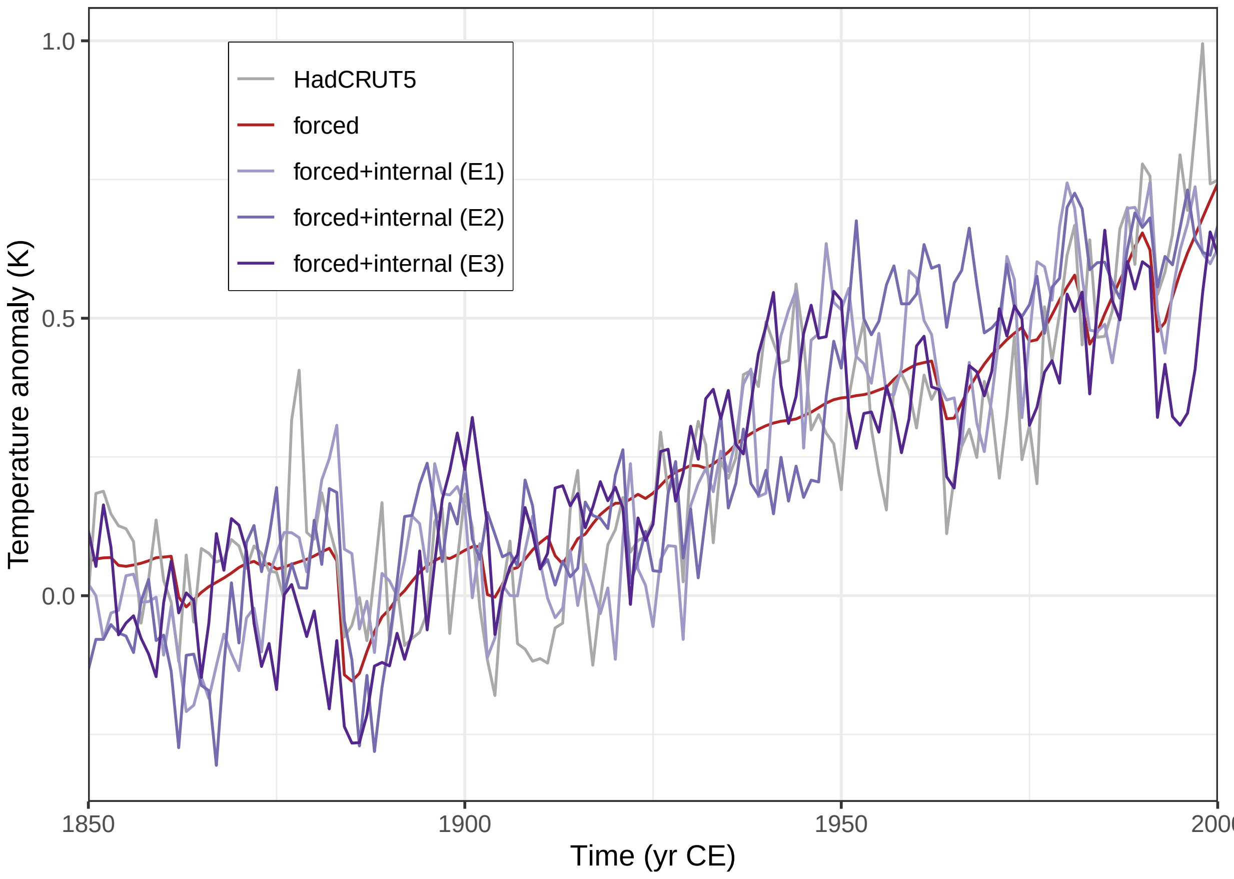
```
library(tidyr)
library(tibble)
library(ggplot2)

#generate samples of internal variability
noise2 <- ClimBayes::gen_noise_from_ebm_fit(3, fit_2box)

#prepare data table for plotting
noise_df2 <- as.data.frame(t(noise2))
noise_df2 <- as.data.frame(apply(noise_df2, 2,
                        function(x) x + fit_2box$posteriors$model_fit$mean))
noise_df2$year <- 1850:2000
```

```r
tidyr::pivot_longer(noise_df2, cols = starts_with("V"),
                    names_to = "realisation", values_to = "y") ->
  noise_df2
fit_df = data.frame(year = 1850:2000, y = fit_2box$posteriors$model_fit$mean) %>%
  tibble::add_column(realisation="2-box")
sol_df = data.frame(year = 1850:2000, y = fit_2box$input_params$y_obs) %>%
  tibble::add_column(realisation="Observation")
my_colors <- RColorBrewer::brewer.pal(6, "Purples")[4:6]

#plot
ggplot2::ggplot(rbind(noise_df2, sol_df, fit_df)) +
  geom_line(aes(x = year, y = y, col=realisation)) +
  labs(x = "Time (yr CE)", y = "Temperature anomaly (K)") +
  scale_x_continuous(expand=c(0,0)) +
  theme_bw() +
  theme(legend.position = "none") +
  scale_color_manual(values=c("Observation"="darkgrey",
                              "2-box"="firebrick",
                              "V1"=my_colors[1],
                              "V2"=my_colors[2],
                              "V3"=my_colors[3]),
                     breaks=c("Observation","2-box","V1","V2","V3"),
                     labels = c("HadCRUT5", "forced", "forced+internal (E1)",
                              "forced+internal (E2)", "forced+internal (E3)")) +
  theme(legend.position=c(0.25,0.8),
        legend.box.background =  element_rect(colour = "black"),
        legend.title= element_blank(),
        legend.margin = ggplot2::margin(t=-0.02,l=0.05,b=0.05,r=0.1, unit='cm')) +
  theme(plot.margin = margin(0,0.5,0,0, "cm"))
```



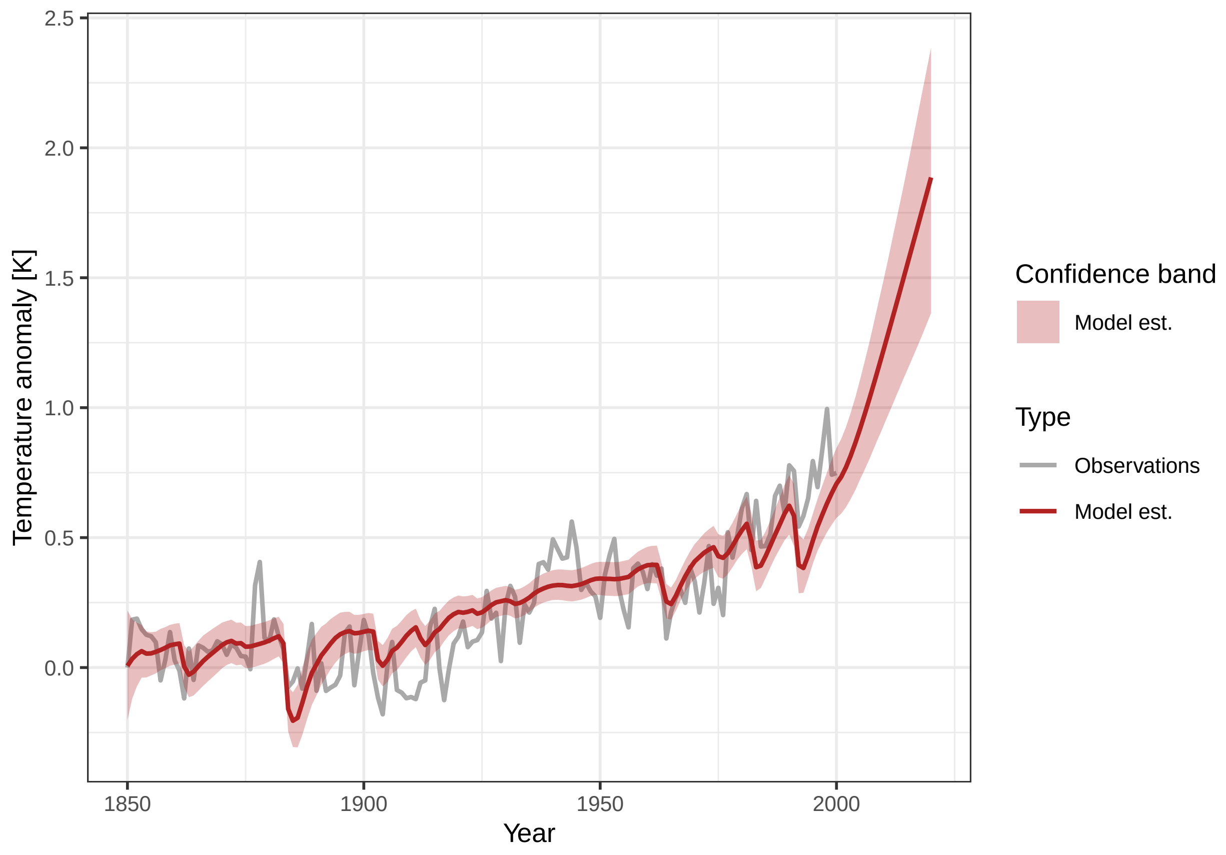
Example for projections

In climate research, observational data might sometimes be missing, but there could be information on the forcing (both for the past, e.g. Milankovitch cycles, and the future, e.g. projected trend in GHG emissions). Using the `ebm_projection` function, the temperature response to these forcings as modeled by the multibox EBM can be easily computed. Yet, it is important to note, that this represents only a single, idealized realization under the assumption of a simple climate model. To compute the temperature response, the forcing values are needed as well as an object of class `ebm_fit` to provide the climate parameters of the model. To show the forcing in the plot below, change `plot_forcing=F` to `plot_forcing=F`.

```
forc_proj = tibble::tibble(year = 2001:2020, forcing = 1:20 / 5)
proj1 <- ebm_projection(fit_1box, forc_proj, cred_int = TRUE)
```

```
## No years for 'forc' supplied. Use entire time series.
```

```
plot(proj1, plot_forcing=F)
```



2.5
2.0
1.5
1.0
0.5
0.0
Temperature anomaly [K]
1850
1900
1950
2000
Year
Confidence band
Model est.
Type
Observations
Model est.