Advanced features of hmmTMB

Théo Michelot

2022-12-12

Contents

We will cover some advanced features of the R package hmmTMB, using the energy price example from the MSwM package (Sanchez-Espigares and Lopez-Moreno (2021)). The vignette "*Analysing time series data with hidden Markov models in hmmTMB*" is a better starting point to learn about the basic functionalities of hmmTMB, and we build on it here. The sections of this document are mostly independent.

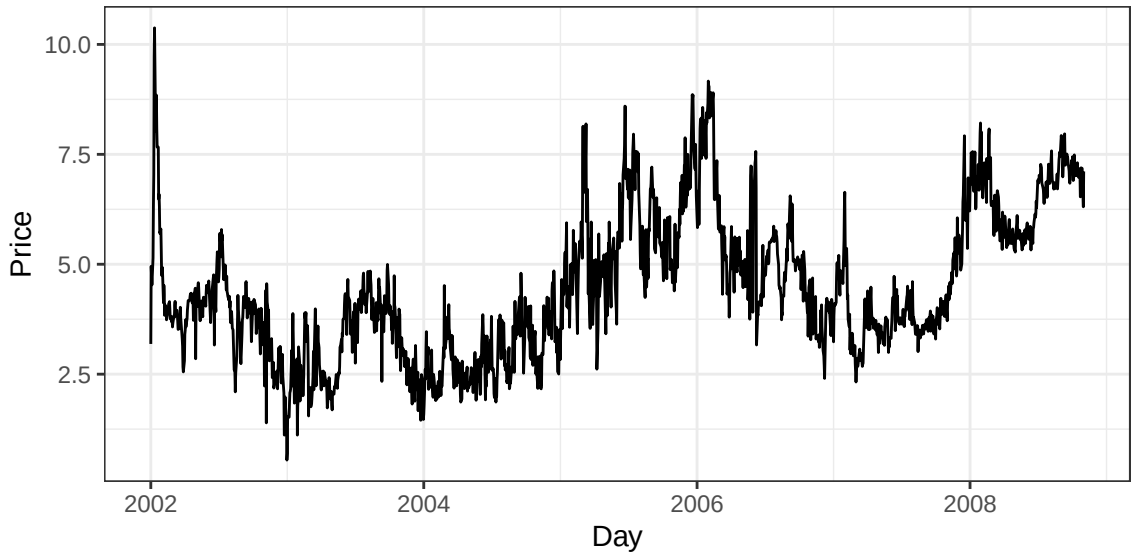
```
# Load packages and color palette
library(ggplot2)
theme_set(theme_bw())
library(hmmTMB)
pal <- hmmTMB:::hmmTMB_cols
```

The data set includes energy prices in Spain between 2002 and 2008, as well as some potential explanatory variables (e.g., raw material prices, financial indices). Refer to `?MSwM::energy`, or to the other vignette, for more detail about the data set.

```
data(energy, package = "MSwM")

# Add time column to data set for plots
day1 <- as.POSIXct("2002/01/01", format = "%Y/%m/%d", tz = "GMT")
day2 <- as.POSIXct("2008/10/31", format = "%Y/%m/%d", tz = "GMT")
days_with_we <- seq(day1, day2, by = "1 day")
which_we <- which(format(days_with_we, '%u') %in% 6:7)
days <- days_with_we[-which_we]
energy$Day <- days

ggplot(energy, aes(Day, Price)) + geom_line()
```



1 Supervised learning

HMMs are most often presented as a method for unsupervised learning: the states are not known a priori, and their definition is data-driven. This is also the default way to use hmmTMB, which is described in other vignettes. In some applications, we might know what the hidden states are for some of the time steps. This could be the case for example in an ecological study where the hidden state is the behavioural state of the animal, if direct observations of the behaviour are available for some period of time. Another example might be a study where the states can easily be identified manually by a human, but only for a subset of the data because of time constraints.

In cases where such information is available, it is valuable to pass it to the model, to help with defining the states. Indeed, in such a supervised setting, the interpretation of the states is known a priori, rather than purely determined by the data. In hmmTMB, known states can be passed to a model through a column named `state` in the data set. If such a column exists, it should include `NA` for time steps when the state is not known, and an integer when the state is known (e.g., either 1 or 2 for a two-state model).

In the energy example, the states are not known a priori but, for the sake of illustration, let's assume that some are. More specifically, let's say that we know that the energy prices after January 1st 2007 are all in state 2 (which might represent "high prices"). We should then create a vector of same length as the data, where all elements up to December 31st 2006 are `NA`, and all elements from January 1st 2007 are 2.

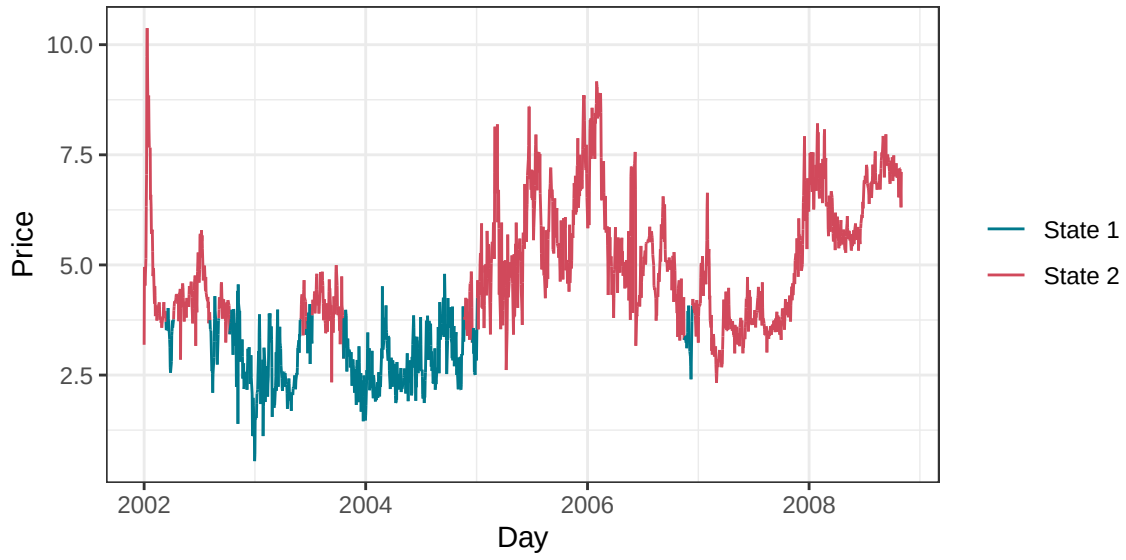
```r
# Create vector of known states
known_states <- rep(NA, nrow(energy))
known_states[which(energy$Day > as.POSIXct("2007/01/01"))] <- 2

# Add to data set
data_supervised <- energy
data_supervised$state <- known_states

# Create model, with data including known states
hid <- MarkovChain$new(data = data_supervised, n_states = 2)
dists <- list(Price = "gamma2")
par0 <- list(Price = list(mean = c(3, 6), sd = c(1, 1)))
obs <- Observation$new(data = data_supervised, n_states = 2,
                       dists = dists, par = par0)
hmm <- HMM$new(hid = hid, obs = obs)

# Fit model
hmm$fit(silent = TRUE)

# Plot prices coloured by most likely state sequence
data_supervised$state <- factor(paste0("State ", hmm$viterbi()))
ggplot(data_supervised, aes(Day, Price, col = state, group = NA)) +
  geom_line() +
  scale_color_manual(values = pal, name = NULL)
```



The plot of the state-coloured time series confirms that the state was fixed to 2 for 2007-2008. This caused the definition of the states to be different from what they would have been without the known states.

2 Using a trained model on new data

In some applications, it might be useful to apply a fitted model to new data, without fitting it again. For example, consider the case where the full data set is too large to fit a model. We could fit a model to a subset of the data, and then use this model to carry out inferences on the full data set (e.g., state decoding). We call "training data" the subset to which the model is fitted, and "new data" the data set to which we wish to apply the fitted model directly. For example, let's say that the training data consists of the first 500 rows of the `energy` data set, and the new data is the rest.

```
# "Training data"
energy_train <- energy[1:500,]
# "New data"
energy_new <- energy[-(1:500),]
```

In hmmTMB, the trick to achieve this is to pass the trained model as `init` when creating the `HMM` object for the new data. When this argument is specified, all parameters of that model are copied into the new model (parameters that the two models have in common). Therefore, this creates an HMM object which stores the new data, but the parameter values estimated from the training data.

```
# Create and fit model to training data
hid_train <- MarkovChain$new(data = energy_train, n_states = 2)
dists <- list(Price = "gamma2")
par0 <- list(Price = list(mean = c(3, 6), sd = c(1, 1)))
obs_train <- Observation$new(data = energy_train, n_states = 2,
                             dists = dists, par = par0)
hmm_train <- HMM$new(hid = hid_train, obs = obs_train)
hmm_train$fit(silent = TRUE)

# Create model for new data, but don't fit
hid_new <- MarkovChain$new(data = energy_new, n_states = 2)
obs_new <- Observation$new(data = energy_new, n_states = 2,
                           dists = dists, par = par0)
hmm_new <- HMM$new(hid = hid_new, obs = obs_new, init = hmm_train)
```

```r
# Check parameters
hmm_new
```

```
########################
## Observation model ##
########################
+ Price ~ gamma2(mean, sd)
  * mean.state1 ~ 1
  * mean.state2 ~ 1
  * sd.state1 ~ 1
  * sd.state2 ~ 1

> Initial observation parameters (t = 1):
           state 1 state 2
Price.mean   2.437   4.181
Price.sd     0.620   0.838

#########################
## State process model ##
#########################
        state 1 state 2
state 1       .      ~1
state 2      ~1       .

> Initial transition probabilities (t = 1):
        state 1 state 2
state 1   0.974   0.026
state 2   0.016   0.984
```

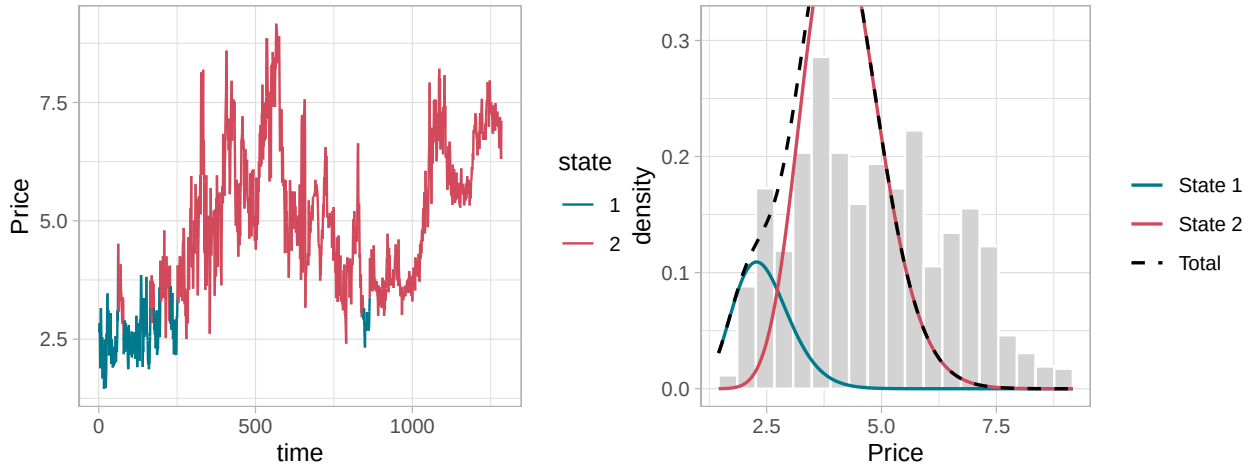
Then, we can use this model without fitting it, for example to get the most likely state sequence for the new data, based on the parameters estimated from the training data. We can also plot some model outputs, like a time series coloured by state, or plots of the new data overlaid with the distributions estimated from the training data. Looking at these outputs actually makes it clear that the method did not work very well in this example: the state-dependent distributions don't fit the new data well. This approach is only useful in cases where patterns in the training data and the new data are similar, so that the states estimates from the former can describe the latter well.

```
# Global state decoding
head(hmm_new$viterbi())
```

```
[1] 1 1 1 1 1 1
```

```
# Plot new data coloured by states (based on trained model)
hmm_new$plot_ts("Price")

# Check whether distributions fit the new data well
hmm_new$plot_dist("Price")
```



3 Fixed parameters

Parameters of the model can be fixed using the argument `fixpar` when creating an `HMM` object. If a parameter is fixed, then it stays fixed at its initial value rather than being estimated. This may be useful in many situations, for example if we know that a given transition probability is zero. It is also possible to specify that two (or more) parameters should be estimated, but with a common value. Under the hood, this functionality uses the argument `map` of `MakeADFun` in TMB, and you can refer to `?TMB::MakeADFun` for more details.

The `fixpar` argument should be a named list with elements from: `obs` (observation parameters), `hid` (transition probabilities), `lambda_obs` (smoothness parameter for observation process), `lambda_hid` (smoothness parameter for hidden process), and `delta0` (initial distribution of hidden process). Each of these entries should be a named vector, in a format similar to that expected by TMB with `map`. Specifically, each element of the vector should be named after a working parameter of the model, which can be found by `hmm$coeff_fe()` or `hmm$lambda()`, and its value should be: (i) `NA` if it is fixed, or (ii) some integer if several parameters have a common value.

The chunk below shows a few examples to illustrate the syntax that should be used, and we describe a couple of examples in more detail in the next subsections.

```
# Intercept for "mean" parameter fixed in state 1
fixpar <- list(obs = c("Price.mean.state1.(Intercept)" = NA))

# Intercept for Pr(1>2) fixed
fixpar <- list(hid = c("S1>S2.(Intercept)" = NA))

# Same intercepts for Pr(S1 -> S2) and Pr(S2 -> S1)
fixpar <- list(hid = c("S1>S2.(Intercept)" = 1,
                       "S2>S1.(Intercept)" = 1))

# Pr(S[1] = 1) fixed
fixpar <- list(delta0 = c("state1" = NA))
```

3.1 Example 1: fixed observation parameters

We first turn to the case where some of the state-dependent observation parameters should be fixed. We fit a 3-state model with a gamma distribution for the energy price, and assume that we know the standard deviations of the three gamma distributions a priori, and therefore fix them to 0.5. To do this, we first need to set the initial standard deviations to 0.5 when creating the `Observation` object.

```
hid <- MarkovChain$new(data = energy, n_states = 3)
dists <- list(Price = "gamma2")
par0 <- list(Price = list(mean = c(2, 5, 7), sd = c(0.5, 0.5, 0.5)))
obs <- Observation$new(data = energy, n_states = 3,
                       dists = dists, par = par0)
```

Then, we use the argument `fixpar` to indicate that they should be kept fixed. The names of the working parameters corresponding to the three standard deviations can be found using `obs$coeff_fe()`.

```
# Find names of working parameters for SDs
obs$coeff_fe()
```

```
                                  [,1]
Price.mean.state1.(Intercept)  0.6931472
Price.mean.state2.(Intercept)  1.6094379
```

```
Price.mean.state3.(Intercept)  1.9459101
Price.sd.state1.(Intercept)   -0.6931472
Price.sd.state2.(Intercept)   -0.6931472
Price.sd.state3.(Intercept)   -0.6931472
```

```
# List of fixed parameters
fixpar <- list(obs = c("Price.sd.state1.(Intercept)" = NA,
                       "Price.sd.state2.(Intercept)" = NA,
                       "Price.sd.state3.(Intercept)" = NA))

# Create and fit model with fixed parameters
hmm <- HMM$new(hid = hid, obs = obs, fixpar = fixpar)
hmm$fit(silent = TRUE)

# Check that standard deviations were kept fixed
hmm
```

```
########################
## Observation model ##
########################
+ Price ~ gamma2(mean, sd)
  * mean.state1 ~ 1
  * mean.state2 ~ 1
  * mean.state3 ~ 1
  * sd.state1 ~ 1
  * sd.state2 ~ 1
  * sd.state3 ~ 1

> Estimated observation parameters (t = 1):
           state 1 state 2 state 3
Price.mean   2.555   4.165   6.553
Price.sd     0.500   0.500   0.500

#########################
## State process model ##
#########################
        state 1 state 2 state 3
state 1       .      ~1      ~1
```

```
state 2      ~1       .      ~1
state 3      ~1      ~1       .

> Estimated transition probabilities (t = 1):
        state 1 state 2 state 3
state 1   0.959   0.041   0.000
state 2   0.020   0.962   0.019
state 3   0.000   0.032   0.968
```

Alternatively, we might want to estimate the standard deviations, but constrain them to have a common value (rather than estimating them as three separate parameters). To do this, we need to set the elements of `fixpar` to some integer (which should be the same for all three standard deviations), rather than `NA`.

```
# List of fixed parameters
fixpar <- list(obs = c("Price.sd.state1.(Intercept)" = 1,
                       "Price.sd.state2.(Intercept)" = 1,
                       "Price.sd.state3.(Intercept)" = 1))

# Create and fit model with fixed parameters
hmm <- HMM$new(hid = hid, obs = obs, fixpar = fixpar)
hmm$fit(silent = TRUE)

# Check that standard deviations have common value
hmm
```

```
########################
## Observation model ##
########################
+ Price ~ gamma2(mean, sd)
  * mean.state1 ~ 1
  * mean.state2 ~ 1
  * mean.state3 ~ 1
  * sd.state1 ~ 1
  * sd.state2 ~ 1
  * sd.state3 ~ 1

> Estimated observation parameters (t = 1):
           state 1 state 2 state 3
```

```
Price.mean   2.693   4.246   6.614
Price.sd     0.719   0.719   0.719

#########################
## State process model ##
#########################
        state 1 state 2 state 3
state 1       .      ~1      ~1
state 2      ~1       .      ~1
state 3      ~1      ~1       .

> Estimated transition probabilities (t = 1):
        state 1 state 2 state 3
state 1   0.978   0.022   0.000
state 2   0.011   0.974   0.014
state 3   0.000   0.025   0.975
```

3.2 Example 2: fixed transition probabilities

Similarly, the transition probabilities can be kept fixed or fixed to a common value using `fixpar`. Note that, due to the multinomial logit link function used for the transition probabilities, it is sometimes impossible to fix a single transition probability. For example, in a 3-state model, fixing `S1>S2.(Intercept)` (the intercept for γ_{12}) does not necessarily fix the value of γ_{12}, because the transition probability also depends on `S1>S3.(Intercept)` (in the denominator of the multinomial logit). There are however three important cases where a transition probability can be fixed:

1. In a 2-state model, fixing the intercept parameter fixes the corresponding transition probability (because each transition probability only depends on one intercept parameter, unlike in models with 3 or more states).
2. If the intercept is fixed to $-\infty$, then the corresponding transition probability is fixed to zero.
3. It is possible to fix a complete row of the transition probability matrix.

The most common application might be a model where some transitions are prohibited, and therefore should have probability fixed to zero. Consider the 3-state model with transition

probability matrix

$$\Gamma = \begin{pmatrix} \gamma_{11} & \gamma_{12} & 0 \\ \gamma_{21} & \gamma_{22} & \gamma_{23} \\ 0 & \gamma_{32} & \gamma_{33} \end{pmatrix},$$

i.e., where transitions cannot occur between states 1 and 3.

This could be implemented by setting the initial values for γ_{13} and γ_{31} to zero, and then using `fixpar` to keep them fixed.

```
# Create Markov chain with initial transition probabilities
tpm0 <- matrix(c(0.9, 0.1, 0,
                 0.1, 0.8, 0.1,
                 0, 0.1, 0.9),
               nrow = 3, byrow = TRUE)
hid <- MarkovChain$new(data = energy, n_states = 3, tpm = tpm0)

# Create observation model
dists <- list(Price = "gamma2")
par0 <- list(Price = list(mean = c(2, 5, 7), sd = c(0.5, 0.5, 0.5)))
obs <- Observation$new(data = energy, n_states = 3,
                       dists = dists, par = par0)

# Find names of working parameters for transition probabilities
hid$coeff_fe()
```

```
                       [,1]
S1>S2.(Intercept) -2.197225
S1>S3.(Intercept)      -Inf
S2>S1.(Intercept) -2.079442
S2>S3.(Intercept) -2.079442
S3>S1.(Intercept)      -Inf
S3>S2.(Intercept) -2.197225
```

```
# List of fixed parameters
fixpar <- list(hid = c("S1>S3.(Intercept)" = NA,
                       "S3>S1.(Intercept)" = NA))

# Create and fit model with fixed parameters
hmm <- HMM$new(hid = hid, obs = obs, fixpar = fixpar)
```

```r
hmm$fit(silent = TRUE)

# Check that transition probabilities were kept fixed
hmm
```

```
########################
## Observation model ##
########################
+ Price ~ gamma2(mean, sd)
  * mean.state1 ~ 1
  * mean.state2 ~ 1
  * mean.state3 ~ 1
  * sd.state1 ~ 1
  * sd.state2 ~ 1
  * sd.state3 ~ 1

> Estimated observation parameters (t = 1):
           state 1 state 2 state 3
Price.mean   2.533   3.893   6.094
Price.sd     0.539   0.441   1.086

#########################
## State process model ##
#########################
        state 1 state 2 state 3
state 1       .      ~1      ~1
state 2      ~1       .      ~1
state 3      ~1      ~1       .

> Estimated transition probabilities (t = 1):
        state 1 state 2 state 3
state 1   0.962   0.038   0.000
state 2   0.023   0.958   0.019
state 3   0.000   0.017   0.983
```

4 Updating a model component

There are functions to modify an existing model without having to create a new model object from scratch.

4.1 Updating model parameters

It is possible to manually update the parameters stored in the different model components, for example to change the initial parameters. We can either change the model parameters themselves (e.g., transition probabilities, observation parameters), or the underlying working parameters. The latter option is mostly useful in cases where the model includes covariates, as this is the only way to change the corresponding effect parameters.

```
# Create model with default transition probabilities
hid <- MarkovChain$new(data = energy, n_states = 2)
hid$tpm()
```

```
, , 1

        state 1 state 2
state 1     0.9     0.1
state 2     0.1     0.9
```

```
# Update transition probabilities
new_tpm <- matrix(c(0.5, 0.5, 0.5, 0.5), nrow = 2)
hid$update_tpm(tpm = new_tpm)
hid$tpm()
```

```
, , 1

        state 1 state 2
state 1     0.5     0.5
state 2     0.5     0.5
```

```
# Create model with covariate and default effect (zero)
hid <- MarkovChain$new(data = energy, n_states = 2, formula = ~Oil)
hid$coeff_fe()
```

```
                       [,1]
S1>S2.(Intercept) -2.197225
S1>S2.Oil          0.000000
```

```
S2>S1.(Intercept) -2.197225
S2>S1.Oil          0.000000
```

```
# Update working parameters
hid$update_coeff_fe(coeff_fe = c(-3, 0.1, -3, -0.1))
hid$coeff_fe()
```

```
                  [,1]
S1>S2.(Intercept) -3.0
S1>S2.Oil          0.1
S2>S1.(Intercept) -3.0
S2>S1.Oil         -0.1
```

In the same way, we can manually modify the parameters (including the working parameters if necessary) of the observation model.

```
# Create observation model with some initial parameters
dists <- list(Price = "gamma2")
par0 <- list(Price = list(mean = c(2, 7), sd = c(0.5, 0.5)))
obs <- Observation$new(data = energy, n_states = 2,
                       dists = dists, par = par0)
obs$par()
```

```
, , 1

           state 1 state 2
Price.mean     2.0     7.0
Price.sd       0.5     0.5
```

```
# Update parameters
new_par <- list(Price = list(mean = c(1, 5), sd = c(0.5, 1)))
obs$update_par(par = new_par)
obs$par()
```

```
, , 1

           state 1 state 2
Price.mean     1.0       5
Price.sd       0.5       1
```

Note that, if these `MarkovChain` and `Observation` objects had been used to create an `HMM`

object, then this would also change the parameters in that `HMM`.

4.2 Updating model formulas

The formulas on any model parameters can also be updated for an existing model, using the `update()` function. This is not an R6 method, so we call it as `update(hmm, ...)` rather than `hmm$update(...)`. First, we create a covariate-free model for illustration.

```
# Create model with no covariates
hid <- MarkovChain$new(data = energy, n_states = 2)
dists <- list(Price = "gamma2")
par0 <- list(Price = list(mean = c(2, 7), sd = c(0.5, 0.5)))
obs <- Observation$new(data = energy, n_states = 2,
                       dists = dists, par = par0)
hmm <- HMM$new(obs = obs, hid = hid)

# Check model formulation
hmm
```

```
########################
## Observation model ##
########################
+ Price ~ gamma2(mean, sd)
  * mean.state1 ~ 1
  * mean.state2 ~ 1
  * sd.state1 ~ 1
  * sd.state2 ~ 1

> Initial observation parameters (t = 1):
           state 1 state 2
Price.mean     2.0     7.0
Price.sd       0.5     0.5

#########################
## State process model ##
#########################
        state 1 state 2
state 1       .      ~1
state 2      ~1       .
```

```
> Initial transition probabilities (t = 1):
        state 1 state 2
state 1     0.9     0.1
state 2     0.1     0.9
```

The arguments of the function `update` are the following:

- `mod` is an `HMM` object.
- `type` refers to the model component that should be changed, either `"hid"` or `"obs"`.
- `i` and `j` are used to indicate which specific formula should be changed. If `type = "hid"`, these are the indices of the transition probability for which the formula should be updated. If `type = "obs"`, `i` is the name of the relevant variable, and `j` the name of the relevant parameter.
- `change` is a template describing how the formula should be updated. See `?update.formula` for more details.
- `fit` is a logical indicating whether the updated model should be fitted or not.

The chunk below illustrates the use of `update`. We first update the hidden state model, to add a linear effect of the covariate `Oil` on the transition probability $\gamma_{12} = \Pr(S_t = 2|S_{t-1} = 1)$, then add a linear effect of the covariate `Oil` on the transition probability $\gamma_{21} = \Pr(S_t = 1|S_{t-1} = 2)$, then add a non-linear effect of `EurDol` on the mean parameter of the `Price` variable.

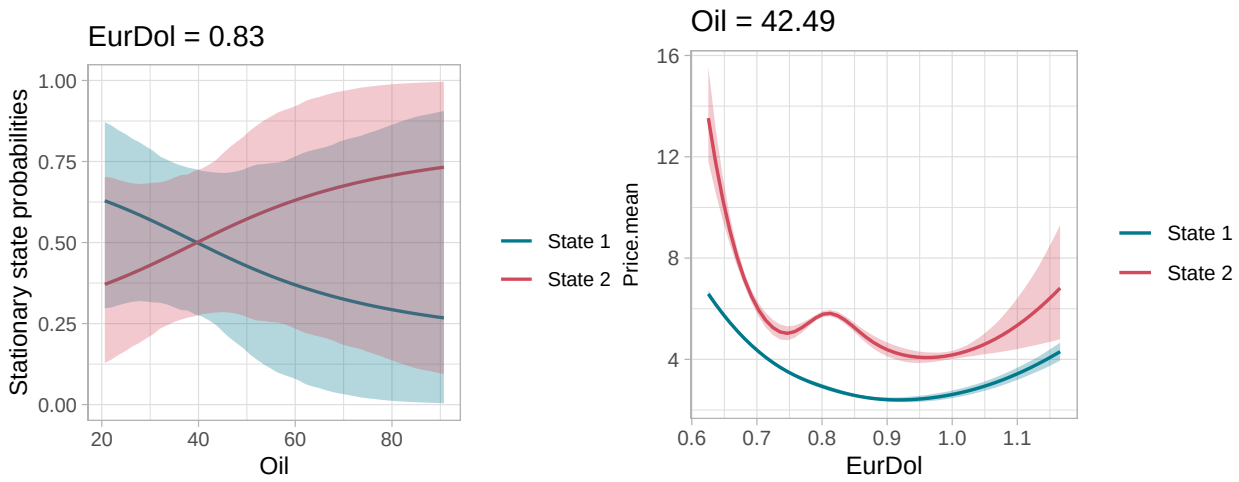
```
# Update formula for Pr(S1 > S2)
hmm2 <- update(object = hmm, type = "hid", i = 1, j = 2,
               change = ~ . + Oil, fit = FALSE)

# Update formula for Pr(S2 > S1)
hmm2 <- update(object = hmm2, type = "hid", i = 2, j = 1,
               change = ~ . + Oil, fit = FALSE)

# Update formula for Price mean
hmm2 <- update(object = hmm2, type = "obs", i = "Price", j = "mean",
               change = ~ . + s(EurDol, k = 5, bs = "cs"),
               fit = TRUE, silent = TRUE)

# Plot covariate effects
hmm2$plot("delta", var = "Oil")
```

```
hmm2$plot("obspar", var = "EurDol", i = "Price.mean")
```



Here, we only decided to fit the final model. However, it can sometimes be a strategy to fit several incrementally more complex models, to ensure that the initial parameter values used for the more complex model are good.

5 Selecting initial parameters

In hmmTMB, the models are fitted using numerical optimisation of the log-likelihood function, based on the `optimx` function. This procedure essentially consists of exploring the parameter space to find the maximum of the log-likelihood, and it requires a starting point from where to start the search. This starting point should be provided by the user, and poorly chosen initial starting values can lead to numerical problems. In particular, if the starting point is too far from the maximum, the optimiser can get stuck in a local maximum of the log-likelihood, and fail to converge to the global maximum. This issue is discussed in more detail in the following vignette of the package moveHMM: https://cran.r-project.org/package=moveHMM/vignettes/moveHMM-starting-values.pdf.

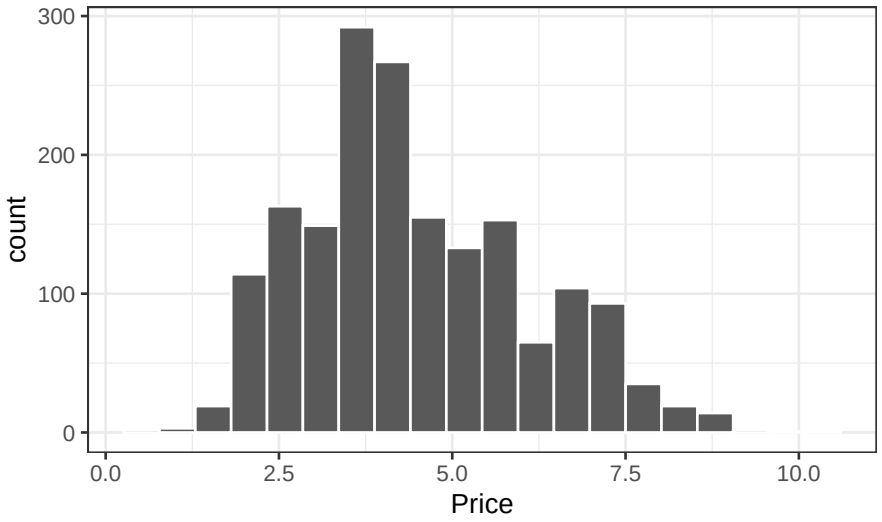
Here, we discuss three strategies for selecting initial parameter values.

5.1 Data visualisation

The closer the initial values are to the global maximum, the easier it is for hmmTMB to converge. So, we want to make an educated guess to choose plausible values, given the data and what we expect the states to be. For this purpose, it is often valuable to plot the response variables, and think about what we might expect the state-dependent distributions to look like.

As an example, consider that we want to model the energy price data by a 2-state model, with gamma distributions, parameterised by a mean and standard deviation (this is the distribution named `"gamma2"` in hmmTMB). We can plot a histogram of the observed prices, and think about how it could be modelled by a mixture of two gamma distributions.

```
ggplot(energy, aes(Price)) + geom_histogram(col = "white", bins = 20)
```



The energy prices are between 0 and 10, so we know that the mean parameters will also be between those two bounds. If we anticipate that one state will capture the lower prices, and one state the higher prices, we can hypothesise that the first mean will be around 3, and the second mean around 6 or so. Similarly, given the spread of the data, it seems plausible that each distribution would have a standard deviation of about 1. So, we could define the initial parameters as a list that contains these values.

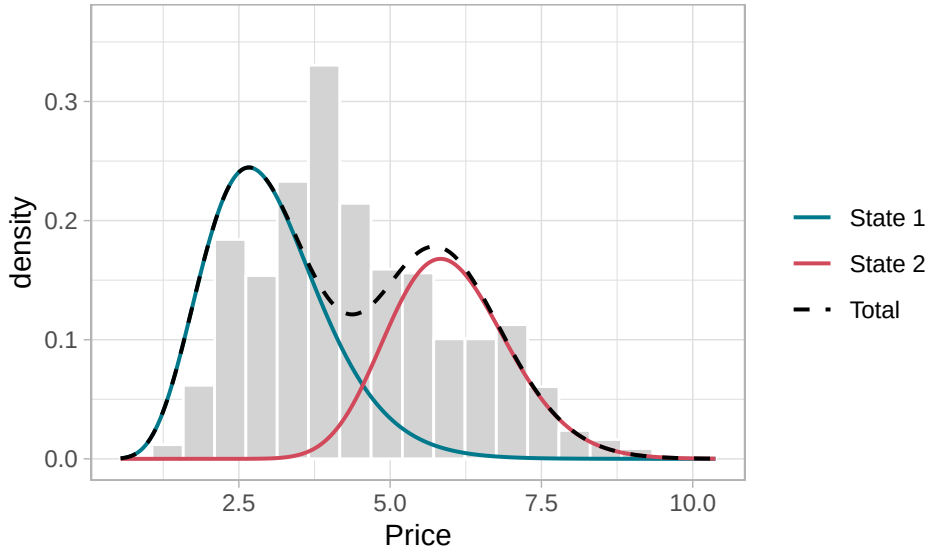
```
par0 <- list(Price = list(mean = c(3, 6), sd = c(1, 1)))
```

In addition to the histogram, it can also be convenient to plot the state-dependent distributions corresponding to a set of starting values, to see if they seem reasonable. This can be done using `plot_dist()` in hmmTMB, and it suggests that the values proposed above were adequate.

```
# Create hidden state process
hid <- MarkovChain$new(data = energy, n_states = 2)
# Create observation model
dists <- list(Price = "gamma2")
par0 <- list(Price = list(mean = c(3, 6), sd = c(1, 1)))
obs <- Observation$new(data = energy, n_states = 2,
                       dists = dists, par = par0)
```

```r
# Create HMM
hmm <- HMM$new(hid = hid, obs = obs)

# Plot histogram of prices, and initial state-dependent distributions
hmm$plot_dist("Price")
```



5.2 Using K-means clustering

Although it is no substitute for thinking critically about what would be plausible initial parameter values, hmmTMB implements a more automated method to suggest initial values for a given model. This is implemented in the function `Observation$suggest_initial()`, which uses the following procedure:

1. Apply K-means clustering to group the observations into N classes (where N is the number of states of the HMM). K-means is a simple clustering algorithm which ignores temporal autocorrelation in the data, but in many cases it will be good enough to roughly identify N coherent groups in the data.
2. For each of the N clusters, estimate parameters for the observation distribution if it were fitted only to the data in that cluster. For example, if we were using normal state-dependent distributions, the parameters for that cluster would be approximated by the empirical mean and standard deviation of the data. In practice, this is done by the function `Dist$parapprox()`.
3. Return a list of the parameters, which can directly be passed to `update_par()`.

The key assumption behind this approach is that the K-means clusters are a good approxi-

mation of the HMM states. When this is not the case, the suggested values might not be very good, so this should be applied with care.

```
# Create observation model
dists <- list(Price = "gamma2")
# These initial values don't matter, we will update them
par0 <- list(Price = list(mean = c(3, 6), sd = c(1, 1)))
obs <- Observation$new(data = energy, n_states = 2,
                       dists = dists, par = par0)

# Get suggested initial parameters
par0 <- obs$suggest_initial()
par0
```

```
$Price
$Price$mean
[1] 3.458191 6.279925

$Price$sd
[1] 0.8299887 1.0096592
```

```
obs$update_par(par = par0)
obs$par()
```

```
, , 1

             state 1  state 2
Price.mean 3.4581913 6.279925
Price.sd   0.8299887 1.009659
```

5.3 From smaller model

A third useful approach to select initial parameter values for complex models is to start from a simple model, and incrementally add model components (e.g., add formulas). In this procedure, the estimated parameters for the simpler model can be used as starting values for the more complex model, which reduces the risk of numerical issues.

Consider that we want to eventually fit a model to the energy prices with the linear effect of oil price on the transition probabilities, and a quadratic relationship between the price mean and the euro-dollar exchange rate. This is a model with many parameters, and there

would be a high chance of convergence issues if we tried to fit it with poorly chosen initial parameter values.

In the code below, we use the argument `init` to create a new `HMM` object with parameters taken from another model. The parameters that are shared are copied from the first model, and parameters that are not shared are kept at their default initial value (e.g., zero for covariate effect parameters).

```
############################
## Model 1: no covariates ##
############################
# Create hidden state process
hid1 <- MarkovChain$new(data = energy, n_states = 2)
# Create observation model
dists <- list(Price = "gamma2")
par0 <- list(Price = list(mean = c(3, 6), sd = c(1, 1)))
obs1 <- Observation$new(data = energy, n_states = 2,
                        dists = dists, par = par0)
# Create HMM
hmm1 <- HMM$new(hid = hid1, obs = obs1)
hmm1$fit(silent = TRUE)

# Estimated (working) parameters
hmm1$coeff_fe()
```

```
$obs
                                     [,1]
Price.mean.state1.(Intercept)  1.2153213
Price.mean.state2.(Intercept)  1.7982319
Price.sd.state1.(Intercept)   -0.1412955
Price.sd.state2.(Intercept)    0.1143659

$hid
                        [,1]
S1>S2.(Intercept) -4.759955
S2>S1.(Intercept) -4.506695
```

```
##############################
## Model 2: covariate on tpm ##
```

```
################################
# Create hidden state process
hid2 <- MarkovChain$new(data = energy, n_states = 2, formula = ~Oil)
# Create observation model
obs2 <- Observation$new(data = energy, n_states = 2,
                        dists = dists, par = par0)
# Create HMM with initial values from hmm1
hmm2 <- HMM$new(hid = hid2, obs = obs2, init = hmm1)
hmm2$coeff_fe()
```

```
$obs
                                     [,1]
Price.mean.state1.(Intercept)  1.2153213
Price.mean.state2.(Intercept)  1.7982319
Price.sd.state1.(Intercept)   -0.1412955
Price.sd.state2.(Intercept)    0.1143659

$hid
                        [,1]
S1>S2.(Intercept) -4.759955
S1>S2.Oil          0.000000
S2>S1.(Intercept) -4.506695
S2>S1.Oil          0.000000
```

```
hmm2$fit(silent = TRUE)
hmm2$coeff_fe()
```

```
$obs
                                     [,1]
Price.mean.state1.(Intercept)  1.2149628
Price.mean.state2.(Intercept)  1.7977311
Price.sd.state1.(Intercept)   -0.1423278
Price.sd.state2.(Intercept)    0.1172540

$hid
                          [,1]
S1>S2.(Intercept) -7.29810266
S1>S2.Oil          0.06405736
```

```
S2>S1.(Intercept) -1.24657266
S2>S1.Oil         -0.06724761
```

```
##############################################
## Model 3: covariates on tpm and obs par ##
##############################################
# Create hidden state process
hid3 <- MarkovChain$new(data = energy, n_states = 2, formula = ~Oil)
# Create observation model
f <- list(Price = list(mean = ~poly(EurDol, 2)))
obs3 <- Observation$new(data = energy, n_states = 2,
                        dists = dists, par = par0, formula = f)
# Create HMM with initial values from hmm1
hmm3 <- HMM$new(hid = hid3, obs = obs3, init = hmm2)
hmm3$coeff_fe()
```

```
$obs
                                          [,1]
Price.mean.state1.(Intercept)        1.2149628
Price.mean.state1.poly(EurDol, 2)1   0.0000000
Price.mean.state1.poly(EurDol, 2)2   0.0000000
Price.mean.state2.(Intercept)        1.7977311
Price.mean.state2.poly(EurDol, 2)1   0.0000000
Price.mean.state2.poly(EurDol, 2)2   0.0000000
Price.sd.state1.(Intercept)         -0.1423278
Price.sd.state2.(Intercept)          0.1172540

$hid
                          [,1]
S1>S2.(Intercept) -7.29810266
S1>S2.Oil          0.06405736
S2>S1.(Intercept) -1.24657266
S2>S1.Oil         -0.06724761
```

```
hmm3$fit(silent = TRUE)
hmm3$coeff_fe()
```

```
$obs
                                          [,1]
```

```
Price.mean.state1.(Intercept)       1.1621909
Price.mean.state1.poly(EurDol, 2)1 -5.6269850
Price.mean.state1.poly(EurDol, 2)2  8.5102715
Price.mean.state2.(Intercept)       1.7066242
Price.mean.state2.poly(EurDol, 2)1 -5.2151098
Price.mean.state2.poly(EurDol, 2)2  3.3671075
Price.sd.state1.(Intercept)        -0.5611061
Price.sd.state2.(Intercept)         0.1704991

$hid
                           [,1]
S1>S2.(Intercept) -4.508134393
S1>S2.Oil         -0.003198406
S2>S1.(Intercept) -2.933638726
S2>S1.Oil         -0.043641114
```

This could also be achieved, in a more concise way, using the `update()` function to add formulas to the model.

6 Accessing the TMB objects

Model fitting in hmmTMB is done using the Template Model Builder (TMB) package (Kristensen et al. (2016)), and the objects created by TMB can be accessed by the user if needed. We consider the 2-state model for the energy data.

```
# Create model
hid <- MarkovChain$new(data = energy, n_states = 2)
dists <- list(Price = "gamma2")
par0 <- list(Price = list(mean = c(3, 6), sd = c(1, 1)))
obs <- Observation$new(data = energy, n_states = 2,
                       dists = dists, par = par0)
hmm <- HMM$new(hid = hid, obs = obs)
# Fit model
hmm$fit(silent = TRUE)
```

6.1 `HMM$tmb_obj()`

The method `tmb_obj()` returns the object created by `TMB::MakeADFun()`, a list with elements that can be used to evaluate the negative log-likelihood function of the model and its

gradient. In hmmTMB, these are passed to `optimx()` for model fitting. Consult the TMB documentation (and particularly `?TMB::MakeADFun`) for more details about what the list components are.

```
# Names of TMB object
names(hmm$tmb_obj())
```

```
 [1] "par"      "fn"       "gr"       "he"       "hessian"  "method"
 [7] "retape"   "env"      "report"   "simulate" "dll"
```

```
# Initial parameters
hmm$tmb_obj()$par
```

```
$coeff_fe_obs
[1] 1.215321

$coeff_fe_obs
[1] 1.798232

$coeff_fe_obs
[1] -0.1412955

$coeff_fe_obs
[1] 0.1143659

$coeff_fe_hid
[1] -4.759955

$coeff_fe_hid
[1] -4.506695

$log_delta0
[1] 14.12881

attr(,"details")
       method   ngatend nhatend hev message
nlminb "nlminb" NA      NA      NA  "none"
attr(,"maximize")
[1] FALSE
```

```
attr(,"npar")
[1] 7
```

```
# Evaluate negative log-likelihood at initial parameters
hmm$tmb_obj()$fn(hmm$tmb_obj()$par)
```

```
[1] 2499.88
```

6.2 HMM$tmb_rep()

After the model is fitted, the function `TMB::sdreport()` is used to get the joint precision matrix of the fixed effect and random effect parameters of the model, which is then used for uncertainty quantification. Mathematical details about how the precision matrix is derived are described in the TMB documentation (`?TMB::sdreport`).

In hmmTMB, this output is available through the `tmb_rep()` function. Printing it shows a table of estimated model parameters and their standard errors.

```
hmm$tmb_rep()
```

```
sdreport(.) result
               Estimate   Std. Error
coeff_fe_obs  1.2153213 8.824388e-03
coeff_fe_obs  1.7982319 8.027100e-03
coeff_fe_obs -0.1412955 2.432127e-02
coeff_fe_obs  0.1143659 2.898627e-02
coeff_fe_hid -4.7599549 3.563663e-01
coeff_fe_hid -4.5066951 3.856447e-01
log_delta0   14.1288105 2.227532e+03
Maximum gradient component: 0.0009864847
```

We can for example extract from this object a vector of estimated fixed effect parameters, and the corresponding covariance matrix. If this model included random effects (or penalised splines), similar objects would be available for the predicted random effects and for the joint precision matrix of the fixed and random effect parameters.

```
# Estimated fixed effect parameters
hmm$tmb_rep()$par.fixed
```

```
coeff_fe_obs coeff_fe_obs coeff_fe_obs coeff_fe_obs coeff_fe_hid coeff_fe_hid
   1.2153213    1.7982319   -0.1412955    0.1143659   -4.7599549   -4.5066951
  log_delta0
```

```
  14.1288105
```

```
# ...and the corresponding covariance matrix
hmm$tmb_rep()$cov.fixed
```

```
              coeff_fe_obs coeff_fe_obs  coeff_fe_obs  coeff_fe_obs
coeff_fe_obs  7.786982e-05 1.521978e-05  9.292244e-05 -3.203772e-05
coeff_fe_obs  1.521978e-05 6.443433e-05  2.795234e-05  6.045490e-06
coeff_fe_obs  9.292244e-05 2.795234e-05  5.915243e-04 -5.279786e-05
coeff_fe_obs -3.203772e-05 6.045490e-06 -5.279786e-05  8.402038e-04
coeff_fe_hid  7.966955e-05 2.042858e-04 -1.269398e-05 -8.218486e-04
coeff_fe_hid  2.011094e-04 3.506594e-04  1.982716e-04 -1.200589e-03
log_delta0    3.095039e-03 5.316369e-03  4.590858e-03 -2.206657e-02
              coeff_fe_hid  coeff_fe_hid    log_delta0
coeff_fe_obs  7.966955e-05  0.0002011094  3.095039e-03
coeff_fe_obs  2.042858e-04  0.0003506594  5.316369e-03
coeff_fe_obs -1.269398e-05  0.0001982716  4.590858e-03
coeff_fe_obs -8.218486e-04 -0.0012005892 -2.206657e-02
coeff_fe_hid  1.269970e-01  0.0172971964  3.646148e-01
coeff_fe_hid  1.729720e-02  0.1487218646  1.093424e-01
log_delta0    3.646148e-01  0.1093424222  4.961898e+06
```

7 Posterior sampling for uncertainty quantification

The parameters that are estimated by TMB in hmmTMB are *working* parameters, i.e., unbounded regression parameters on some link scale, rather than the HMM parameters directly. This is because models are fitted using unbounded optimisation, and also to allow for covariate dependence on the HMM parameters. So, TMB provides a covariance matrix and standard errors for the working parameters, but we are more often interested in quantifying the uncertainty on the HMM parameters themselves (possibly for some fixed covariate values). There are several approaches to uncertainty quantification for transformed quantities, e.g., the delta method.

The main method implemented in hmmTMB is to sample from the approximate distribution of the maximum likelihood estimators. Under maximum likelihood theory (and asymptotic conditions), the estimators follow a multivariate normal distribution centred on the true parameter values, with a covariance matrix which can be approximated by the inverse of the Hessian of the negative log-likelihood. Based on this result, we can use the following workflow:

1. Simulate K replicates from sampling distribution of working parameter estimators, $\theta_1, \theta_2, \ldots, \theta_K$ (where K is a large number). Here, each θ_k is a vector that includes both fixed effect and random effect parameters of the model.
2. Transform each θ_k to obtain the parameter(s) of interest, $g(\theta_k)$. This could mean applying an inverse link function, for example.
3. Use the $g(\theta_k)$ for uncertainty quantification. For example, to obtain an approximate 95% confidence interval of $g(\theta)$, we would define the lower and upper bounds as the 0.025 and 0.975 quantiles of the $g(\theta_k)$, respectively. The approximation will get better as K increases.

This procedure is implemented in the function `predict()` in hmmTMB, which can return confidence intervals for the HMM parameters (for a given data frame of covariate values). Here, we describe lower-level functions that can be used to generate confidence intervals for custom functions g, for example. For illustration, we create a model with the linear effect of `Oil` on the transition probabilities.

```
# Create model
hid <- MarkovChain$new(data = energy, n_states = 2, f = ~Oil)
dists <- list(Price = "gamma2")
par0 <- list(Price = list(mean = c(3, 6), sd = c(1, 1)))
obs <- Observation$new(data = energy, n_states = 2,
                       dists = dists, par = par0)
hmm <- HMM$new(hid = hid, obs = obs)
# Fit model
hmm$fit(silent = TRUE)
```

The function `post_coeff()` implements the first step in the workflow above, and generates posterior samples for the working parameters. Specifically, it returns a matrix with one row for each replicate, and one column for each parameter. The column names help identify the parameters; e.g., in this example, `coeff_fe_obs` are fixed effect parameters for the observation model, `coeff_fe_hid` are fixed effect parameters for the transition probabilities hidden state model, and `log_delta0` are the parameters for the initial distribution of the state process.

```
hmm$post_coeff(n_post = 2)
```

```
     coeff_fe_obs coeff_fe_obs coeff_fe_obs coeff_fe_obs coeff_fe_hid
[1,]     1.219630     1.808613   -0.1043305    0.1440831    -7.922866
[2,]     1.220964     1.796139   -0.1339997    0.0951040    -8.551911
```

```
     coeff_fe_hid coeff_fe_hid coeff_fe_hid log_delta0
[1,]   0.08194844   -1.7239102  -0.06172204  2716.4423
[2,]   0.09399442   -0.5470578  -0.08506730   446.5483
```

The functions `post_linpred()` and `post_fn()` build on `post_coeff()` to generate posterior samples for the linear predictor of each model parameter, and for some user-defined function of the linear predictor, respectively.

References

Kristensen, Kasper, Anders Nielsen, Casper W. Berg, Hans Skaug, and Bradley M. Bell. 2016. "TMB: Automatic Differentiation and Laplace Approximation." *Journal of Statistical Software* 70 (5): 1–21. https://doi.org/10.18637/jss.v070.i05.

Sanchez-Espigares, Josep A., and Alberto Lopez-Moreno. 2021. *MSwM: Fitting Markov Switching Models*. https://CRAN.R-project.org/package=MSwM.