General dependence structures in hmmTMB

Théo Michelot

2024-04-30

Contents

1 Introduction

The package hmmTMB can be used to implement a wide range of extensions of hidden Markov models (HMMs), and we describe a few of them here. This document is not intended to be introductory, and one of the other vignettes might be a better place to start learning about hmmTMB. All dependence structures that we mention in this document are described in more detail by Zucchini, MacDonald, and Langrock (2017) (in particular Chapters 10 and 12).

Do I really need a more complex model?

This vignette showcases the flexibility of hmmTMB to create complex models, and those come with increased risk of running into numerical problems. This may include prohibitive memory or computing requirements for models with very many parameters or, more often, numerical instability in the estimation. Complex models are also more difficult to interpret. It is therefore useful to carefully consider the necessity for increased complexity, for a given data set and research question.

```
library(ggplot2)
theme_set(theme_bw())
library(hmmTMB)
set.seed(534)
```

2 Autoregressive hidden Markov models

The standard assumptions of HMMs can be modified to allow the observation Z_t to depend on the previous observation Z_{t-1}, rather than only on the state S_t. This relaxes the usual conditional independence assumption, and creates additional correlation between successive observations. Such models can be implemented in hmmTMB by including a lagged version of the observed variable as a covariate on the state-dependent observation parameters. This approach can be used to implement random walks with state-switching variance (and/or drift), or autoregressive (AR) models with state-switching parameters, for example.

2.1 State-switching Gaussian random walk

Consider the following Gaussian random walk with state-dependent variance,

$$Z_t = Z_{t-1} + \varepsilon_t, \quad \varepsilon_t \sim N(0,\ \sigma^2_{S_t}),$$

or, equivalently,

$$Z_t \mid \{S_t = j, Z_{t-1} = z_{t-1}\} \sim N(z_{t-1},\ \sigma_j^2).$$

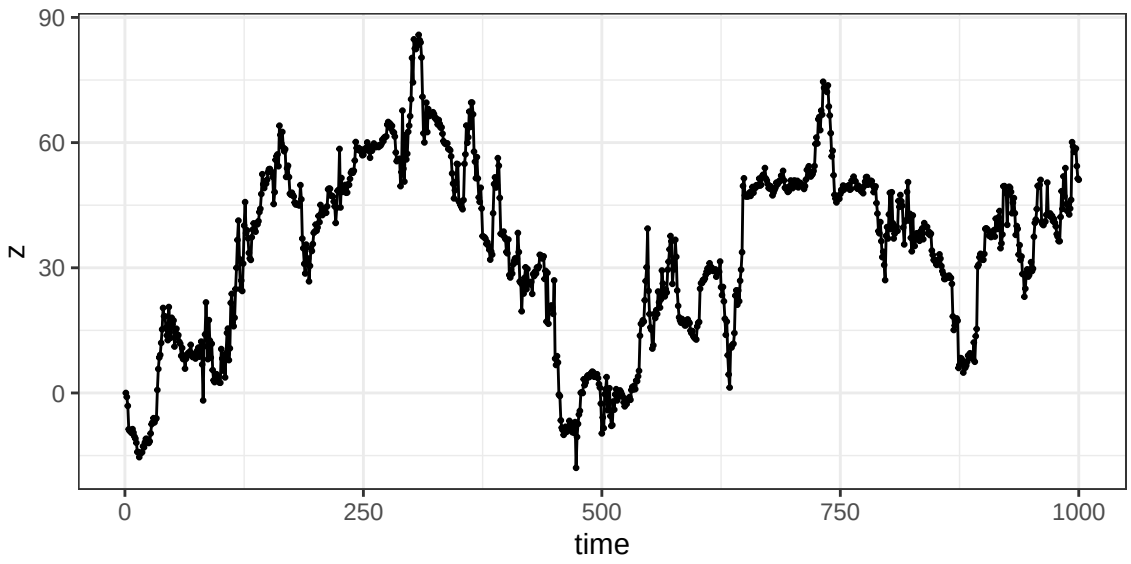
This is an HMM where:

- the observation distributions are normal;
- the mean of the normal distribution depends on the lagged sequence of observations;
- the regression coefficient associated with this lagged covariate is fixed to 1 (i.e., there is no coefficient in front of Z_{t-1}).

```
# Have a look at the data
head(data)
```

```
      z z_lag time
1  0.00    NA    1
2 -0.96  0.00    2
3 -3.09 -0.96    3
4 -8.68 -3.09    4
5 -9.12 -8.68    5
6 -9.13 -9.12    6
```

```
ggplot(data, aes(time, z)) + geom_line() + geom_point(size = 0.5)
```

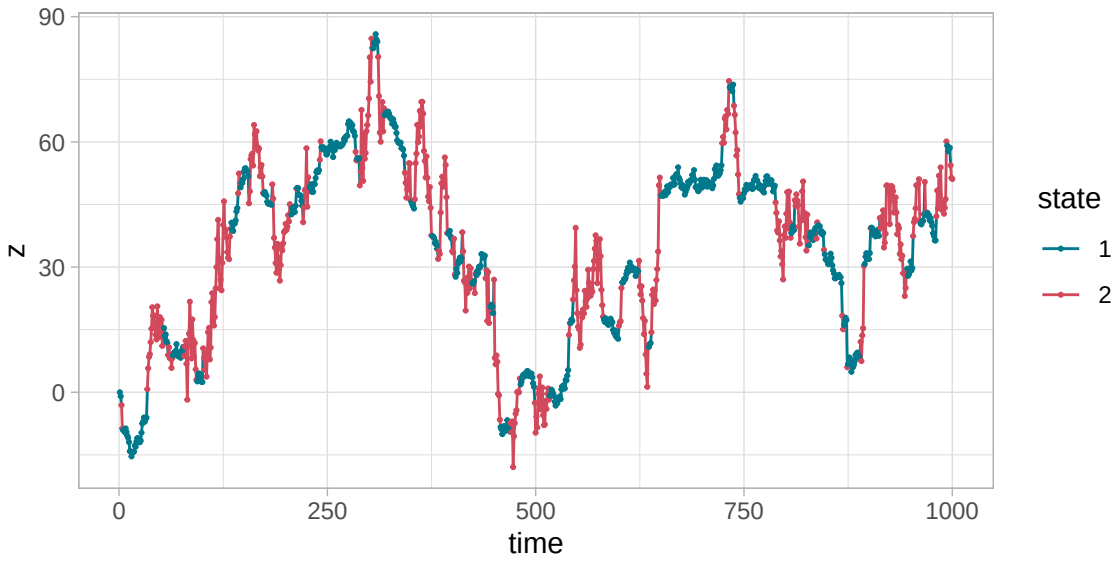



```
# Hidden process model
hid <- MarkovChain$new(data = data, n_states = 2)
```

```
# Observation model
dists <- list(z = "norm")
par0 <- list(z = list(mean = c(0, 0), sd = c(0.5, 10)))
f <- list(z = list(mean = ~ z_lag - 1))
obs <- Observation$new(data = data, dists = dists,
                       formulas = f, par = par0)
obs$update_coeff_fe(c(1, 1, log(1), log(5)))

# Parameter constraints
fixpar <- list(obs = c("z.mean.state1.z_lag" = NA,
                       "z.mean.state2.z_lag" = NA))

# HMM
hmm <- HMM$new(obs = obs, hid = hid, fixpar = fixpar)
hmm$fit(silent = TRUE)

obs$par()
```

```
, , 1

       state 1 state 2
z.mean    0.00     0.0
z.sd      0.99     5.1
```

```
hmm$plot_ts("z") + geom_point(size = 0.4)
```



2.2 Other formulations

We briefly describe a few other models that can be implemented with minimal change to the code above.

2.2.1 Random walk with drift

$$Z_t \mid \{S_t = j, Z_{t-1} = z_{t-1}\} \sim N(\alpha_{0j} + z_{t-1},\ \sigma_j^2)$$

The only difference with the random walk without drift, is that an intercept should be included in the formula for the mean. That is, `mean = ~ z_lag - 1` is replaced by `mean = ~ z_lag`. The intercept coefficient is α_{0j}, and it measures the expected drift in the process over one time step.

2.2.2 AR(1) model

$$Z_t \mid \{S_t = j, Z_{t-1} = z_{t-1}\} \sim N(\alpha_{1j} z_{t-1},\ \sigma_j^2)$$

The first-order autoregressive ("AR(1)") model is widely used, in particular with $|\alpha_{1j}| < 1$ (which leads to a stationary process). In this case, there is no intercept coefficient in the definition of the mean parameter of the normal distribution, but there is a slope coefficient for the lagged covariate. We use the same formula as for the random walk, `mean = ~ z_lag - 1`, but we do not set the parameter constraints through `fixpar`, so that α_{11} and α_{12} are estimated (rather than fixed to 1).

2.2.3 Higher-order models

Higher-order AR models could similarly be implemented by including additional lagged variables as covariates, e.g., observed variable lagged by two time intervals for AR(2),

$$Z_t \mid \{S_t = j, Z_{t-1} = z_{t-1}, Z_{t-2} = z_{t-2}\} \sim N(\alpha_{1j} z_{t-1} + \alpha_{2j} z_{t-2},\ \sigma_j^2)$$

Another option would be to include something like the difference between the two previous observations as a covariate on the mean, to obtain a correlated random walk:

$$Z_t \mid \{S_t = j, Z_{t-1} = z_{t-1}, Z_{t-2} = z_{t-2}\} \sim N(z_{t-1} + (z_{t-1} - z_{t-2}),\ \sigma_j^2)$$

3 Semi-Markov models

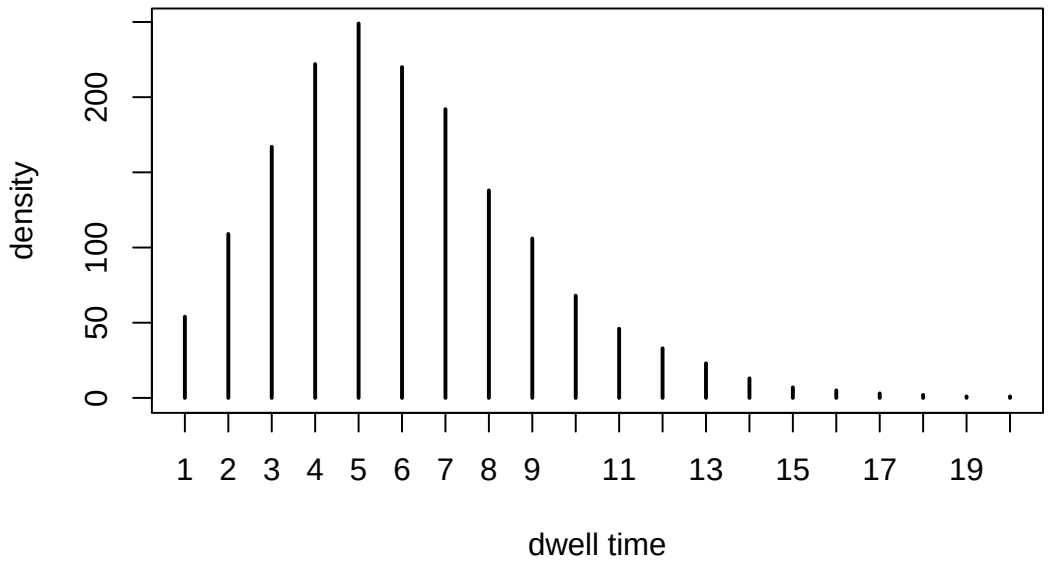
Hidden semi-Markov models (HSMMs) extend HMMs to allow for flexible distributions of dwell times (also known as holding times, i.e., time spent in a state before switching). In a

regular HMM, the dwell times follow a geometric distribution, whereas the distribution is modelled explicitly in an HSMM (e.g., using a Poisson or negative binomial distribution). Langrock and Zucchini (2011) showed that an HSMM can be approximated by an HMM with a particular structure, and we use this approach here. We simulate some data from a hidden semi-Markov model using a negative binomial distribution of dwell times, and we then fit a regular HMM and an (approximate) HSMM using hmmTMB.

3.1 Simulate data

We first simulate a sequence of states from a semi-Markov chain. To do this, we initialise the first state at 1, and we then generate dwell times from a zero-truncated negative binomial distribution with size parameter 10 and mean parameter 5. These dwell times define the number of time steps that the state process stays in the current state before switching to the other one. A density plot of the dwell times shows that their distribution is clearly not geometric (as geometric mass functions are decreasing with a mode at 1).

```
# Number of observations
n <- 1e4
# Initialise state sequence
s <- NULL
state <- 1
# Initialise vector of dwell times
dwells <- NULL
i <- 1
# Loop until the dwell times add up to n or more
while(sum(dwells) < n) {
    dwells[i] <- rnbinom(n = 1, size = 10, mu = 6)
    if(dwells[i] > 0) {
        s <- c(s, rep(state, dwells[i]))
        state <- 3 - state
        i <- i + 1
    }
}
s <- s[1:n]
plot(table(dwells), xlab = "dwell time", ylab = "density")
```



Using the state sequence, we generate the observations from state-dependent normal distributions: $N(-5, 5^2)$ in state 1, and $N(5, 5^2)$ in state 2.

```
# Observation process
z <- rnorm(n, mean = c(-5, 5)[s], sd = c(3, 3)[s])
data <- data.frame(z = z)
```

3.2 Hidden Markov model analysis

We fit a regular HMM to investigate its capacity to capture features of the simulated data set. You can refer to another vignette for details about the syntax used here.

```
# Hidden state model
hid <- MarkovChain$new(data = data, n_states = 2)
dists <- list(z = "norm")
par0 <- list(z = list(mean = c(-1, 1), sd = c(5, 5)))
# Observation model
obs <- Observation$new(data = data, dists = dists, par = par0)
# HMM
hmm <- HMM$new(obs = obs, hid = hid)
hmm$fit(silent = TRUE)
```

We extract a few variables from the fitted model:

- The state-dependent observation parameters were estimated well.
- The state sequence obtained from the Viterbi algorithm is very similar to the true state sequence (98% agreement).
- The mean dwell time was slightly underestimated (estimate = 5.5, truth = 6).

```
# Estimated parameters
hmm$par()$obspar[,,1]
```

```
       state 1 state 2
z.mean    -5.0       5
z.sd       2.9       3
```

```
# Proportion of correctly estimated states
vit <- hmm$viterbi()
sum(vit == s)/n
```

```
[1] 0.98
```

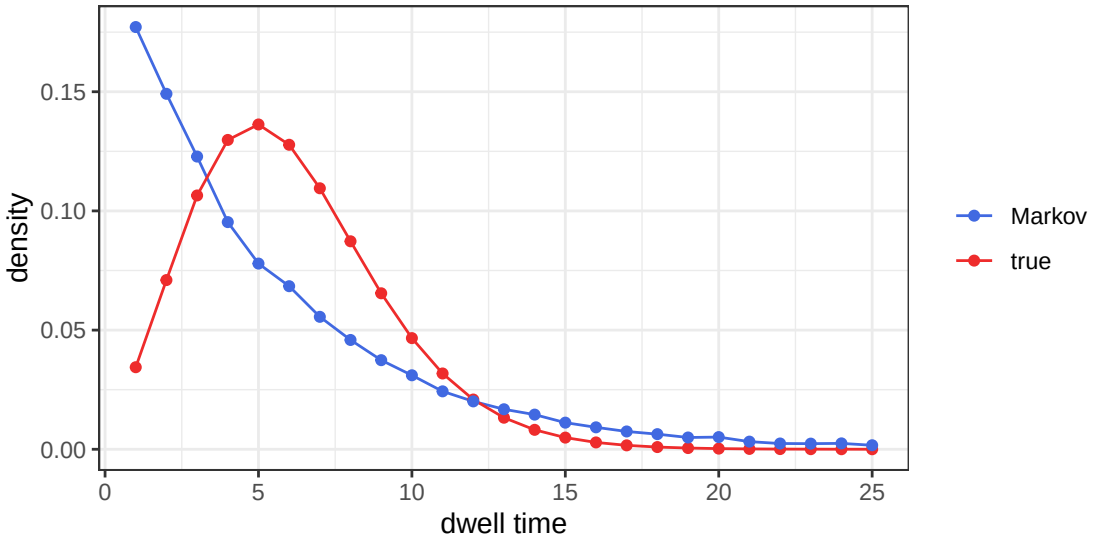
```
# Mean dwell time
1/hid$tpm()[2,1,]
```

```
[1] 5.6
```

These all seem to indicate that the model captured many important features of the data well: it was able to identify and distinguish the two states. The main aim of HSMMs is to capture the dwell time distribution, so let's check whether this was captured by the simple HMM first. One simple way to do this is through simulation: we can simulate from the `MarkovChain` object directly, and then measure the dwell times in the simulated sequence.

```
# Simulate Markov chain
sim_mc <- hid$simulate(n = 1e5, silent = TRUE)

# Plot over a grid of dwell times
x <- 1:25
# Zero-truncated negative binomial distribution
dens_true <- dnbinom(x, size = 10, mu = 6)/(1 - dnbinom(0, size = 10, mu = 6))
# Distribution of dwell times in Markov sequence
dens_markov <- sapply(x, function(i)
    length(which(rle(sim_mc)$lengths == i))/length(rle(sim_mc)$lengths))
```

```r
# Compare the two distributions
df <- data.frame(dwell_time = x,
                 dens = c(dens_true, dens_markov),
                 type = rep(c("true", "Markov"), each = length(x)))
ggplot(df, aes(dwell_time, dens, col = type)) +
    geom_line() + geom_point() +
    labs(x = "dwell time", y = "density", color = NULL) +
    scale_color_manual(values = c("royalblue", "firebrick2"))
```



As expected, the dwell times in the state sequence simulated from the Markov process have a strictly decreasing distribution, with a mode at 1. (This is the geometric distribution.) It does not capture the true negative binomial distribution well.

3.3 (Approximate) hidden semi-Markov model analysis

We do not describe the methodological approach of Langrock and Zucchini (2011) in detail here, and refer to that paper for more information. The basic idea is that an HSMM can be approximated by an HMM with a higher-dimensional state space. Each HSMM state is made up of some number $m > 1$ of HMM states, and a particular structure is used for the resulting transition probability matrix. Larger values of m lead to more flexibility in the dwell time distribution, but also increased computational cost. We apply this method with $m = 6$, leading to a 12-state HMM. There is a very large number of parameters in this model, but many of them will be constrained to be either fixed to zero, or to share a common value, such that the number of estimated parameters is much lower.

```
# Transition matrix
n_states <- 2
m <- 6
B1 <- matrix(0, nrow = m, ncol = m)
B1[cbind(1:m, c(2:m, m))] <- 0.9
B2 <- matrix(c(rep(0.1, m), rep(0, m*(m-1))), ncol = m)
tpm0 <- rbind(cbind(B1, B2), cbind(B2, B1))
# Format as sparse matrix to replace 0 by . in printed output
Matrix::formatSparseM(tpm0)
```

```
 [1,] ".  " "0.9" ".  " ".  " ".  " ".  " "0.1" ".  " ".  " ".  " ".  " ".  "
 [2,] ".  " ".  " "0.9" ".  " ".  " ".  " "0.1" ".  " ".  " ".  " ".  " ".  "
 [3,] ".  " ".  " ".  " "0.9" ".  " ".  " "0.1" ".  " ".  " ".  " ".  " ".  "
 [4,] ".  " ".  " ".  " ".  " "0.9" ".  " "0.1" ".  " ".  " ".  " ".  " ".  "
 [5,] ".  " ".  " ".  " ".  " ".  " "0.9" "0.1" ".  " ".  " ".  " ".  " ".  "
 [6,] ".  " ".  " ".  " ".  " ".  " "0.9" "0.1" ".  " ".  " ".  " ".  " ".  "
 [7,] "0.1" ".  " ".  " ".  " ".  " ".  " "0.9" ".  " ".  " ".  " ".  " ".  "
 [8,] "0.1" ".  " ".  " ".  " ".  " ".  " ".  " "0.9" ".  " ".  " ".  " ".  "
 [9,] "0.1" ".  " ".  " ".  " ".  " ".  " ".  " ".  " "0.9" ".  " ".  " ".  "
[10,] "0.1" ".  " ".  " ".  " ".  " ".  " ".  " ".  " ".  " "0.9" ".  " ".  "
[11,] "0.1" ".  " ".  " ".  " ".  " ".  " ".  " ".  " ".  " ".  " ".  " "0.9"
[12,] "0.1" ".  " ".  " ".  " ".  " ".  " ".  " ".  " ".  " ".  " ".  " "0.9"
```

When creating the `MarkovChain` object, we also need to specify reference indices for the transition probability matrix. These are the indices of the elements of the matrix that are not estimated, but deduced from other elements based on row constraints (entries of each row sum to 1). By default, the references are the diagonal elements, but this makes it impossible to fix them to zero, so here we choose one non-zero transition probability on each row as the reference.

```
# Indices of reference transition probabilities
ref <- apply(tpm0, 1, which.max)
ref
```

```
 [1]  2  3  4  5  6  6  8  9 10 11 12 12
```

```
# Hidden state model
hid2 <- MarkovChain$new(data = data, n_states = n_states * m, tpm = tpm0,
                        ref = ref, initial_state = "stationary")
```

We are using a 12-state HMM, so we need to specify 12 initial means and 12 initial standard deviations to create the `Observation` model. However, the idea is that the first 6 states capture state 1 in the HSMM, and the last 6 capture state 2, so a lot of parameters are shared. This is reflected in our choice of initial values.

```
# Observation model
dists <- list(z = "norm")
par0 <- list(z = list(mean = c(rep(-1, m), rep(1, m)),
                      sd = rep(5, n_states * m)))
par0
```

```
$z
$z$mean
 [1] -1 -1 -1 -1 -1 -1  1  1  1  1  1  1

$z$sd
 [1] 5 5 5 5 5 5 5 5 5 5 5 5
```

```
obs2 <- Observation$new(data = data, dists = dists, par = par0)
```

Many parameter constraints are required in this model, and these can be enforced through the `fixpar` argument of `HMM$new()`. This should be a list with elements `hid` and `obs`, corresponding to constraints on the hidden state and observation models, respectively. Each element is a named vector filled with NAs (for parameters that should be fixed to their initial value) and integers (for parameters that should be estimated to a shared value). This uses the `map` argument of `TMB::MakeADFun()`, and the TMB documentation provides some details; also see the hmmTMB vignette on "Advanced features".

For the transition probability constraints, we:

1. Find the estimated parameters corresponding to transition probabilities fixed to zero. These are intercepts on the link scale, and they are equal to `-Inf` here (which is the result of applying the multinomial logit link function to zero)
2. Create a vector with one `NA` for each such estimated parameter. This will let TMB know that those parameters should be fixed to their initial value, i.e., the corresponding transition probabilities will be fixed to zero.
3. Name the vector of `NA`s after the estimated parameters (where the names can be found with `MarkovChain$coeff_fe()`).

```
# Parameter constraints for tpm
fix_hid <- rep(NA, length(which(is.infinite(hid2$coeff_fe()))))
names(fix_hid) <- rownames(hid2$coeff_fe())[which(is.infinite(hid2$coeff_fe()))]
```

For the observation parameter constraints, we:

1. Create a vector of integers, with m 1s (for mean in state 1), m 2s (mean in state 2), m 3s (SD in state 1), and m 4s (SD in state 2). This will let TMB know that the m first parameters should be estimated to a common value, and likewise the next m parameters, and so on.
2. Get the parameter names from `Observation$coeff_fe()`.

```
# Parameter constraints for observation model
fix_obs <- rep(1:4, each = m)
names(fix_obs) <- rownames(obs2$coeff_fe())

# List of all parameter constraints, to pass to HMM object
fixpar <- list(hid = fix_hid, obs = fix_obs)
```

We can now combine the `MarkovChain` and `Observation` models into an `HMM` object, and pass `fixpar` to specify parameter constraints. Although there are many fewer parameters to estimate than in a regular 12-state HMM, model fitting takes longer than for the simple 2-state HMM (about 5-6 times slower on my laptop).

```
hmm2 <- HMM$new(obs = obs2, hid = hid2, fixpar = fixpar)
hmm2$fit(silent = TRUE)
```

The estimated parameters are very close to the truth. To compare the Viterbi states to the true state sequence, we agglomerate the first 6 states into state 1, and the last 6 states into state 2. The states were recovered almost perfectly.

```
# Observation parameters
hmm2$par()$obs[, c(1, m + 1), 1]
```

```
       state 1 state 7
z.mean      -5       5
z.sd         3       3
```

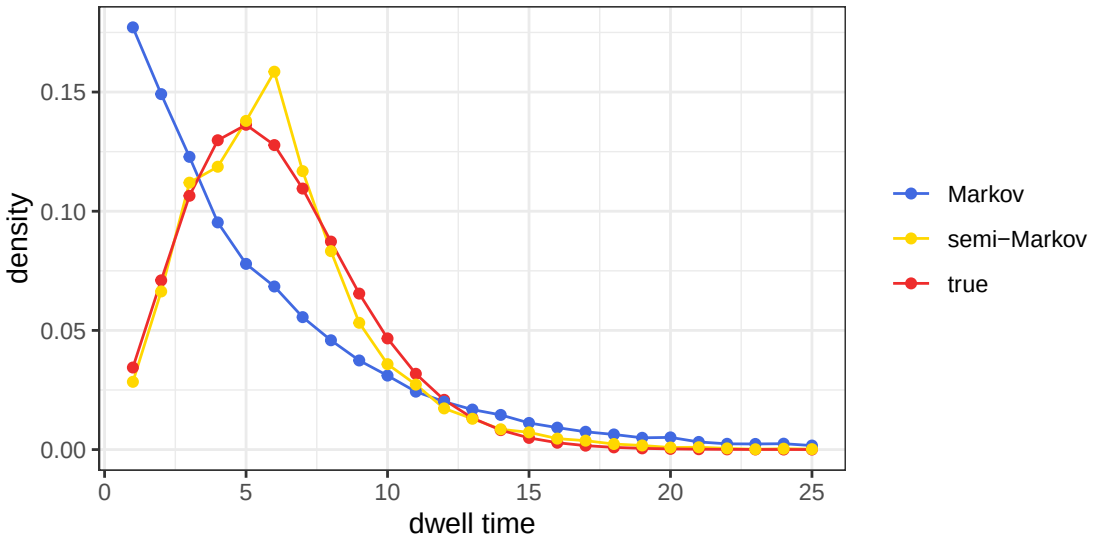
```
# Check decoded states
vit2 <- hmm2$viterbi()
vit2[which(vit2 %in% 1:m)] <- 1
```

```r
vit2[which(vit2 %in% (m + 1):(2 * m))] <- 2
sum(vit2 == s)/n
```

```
[1] 0.98
```

The last check is to compare the distribution of dwell times in the Markov model and the semi-Markov model. We use simulations again, then agglomerate the 12 states into 2 states, and measure dwell times in the simulated sequence. The distribution of dwell times is a much better approximation of the true negative binomial distribution; in particular, it captures its non-monotonic shape (with a mode around 5). Increasing m would improve the approximation, but memory is likely to be a bottleneck on many machines, as the size of model matrices will quickly increase with m.

```r
# Simulate from Markov chain
sim_mc2 <- hid2$simulate(n = 1e5, silent = TRUE)
sim_mc2[which(sim_mc2 %in% 1:m)] <- 1
sim_mc2[which(sim_mc2 %in% (m + 1):(2 * m))] <- 2

# Plot different dwell time distributions
dens_semi_markov <- sapply(x, function(i)
    length(which(rle(sim_mc2)$lengths == i))/length(rle(sim_mc2)$lengths))

df <- rbind(df, data.frame(dwell_time = x,
                           dens = dens_semi_markov,
                           type = "semi-Markov"))

ggplot(df, aes(dwell_time, dens, col = type)) +
    geom_line() + geom_point() +
    labs(x = "dwell time", y = "density", color = NULL) +
    scale_color_manual(values = c("royalblue", "gold", "firebrick2"))
```



4 Higher-order Markov process

The most common HMM formulation relies on a *first-order* Markov state process, where the state S_t only depends on the previous state S_{t-1}. In a k-th order Markov process, the state S_t depends on $\{S_{t-1}, S_{t-2}, \ldots, S_{t-k}\}$, making it possible to account for stronger "memory" in the state process. These higher-order Markov processes can be written as a regular Markov process with an augmented state space. Here, we focus on a second-order model for simplicity.

We denote as $\gamma_{ijk} = \Pr(S_t = k | S_{t-1} = j, S_{t-2} = i)$ the transition probabilities of a second-order Markov process. A 2-state second-order model can be represented by a 4-state process $(\tilde{S}_t)$ on pairs of states, i.e.,

$$\tilde{S}_t = 1 \text{ if } S_t = 1 \text{ and } S_{t-1} = 1$$
$$\tilde{S}_t = 2 \text{ if } S_t = 2 \text{ and } S_{t-1} = 1$$
$$\tilde{S}_t = 3 \text{ if } S_t = 1 \text{ and } S_{t-1} = 2$$
$$\tilde{S}_t = 4 \text{ if } S_t = 2 \text{ and } S_{t-1} = 2$$

with transition probability matrix

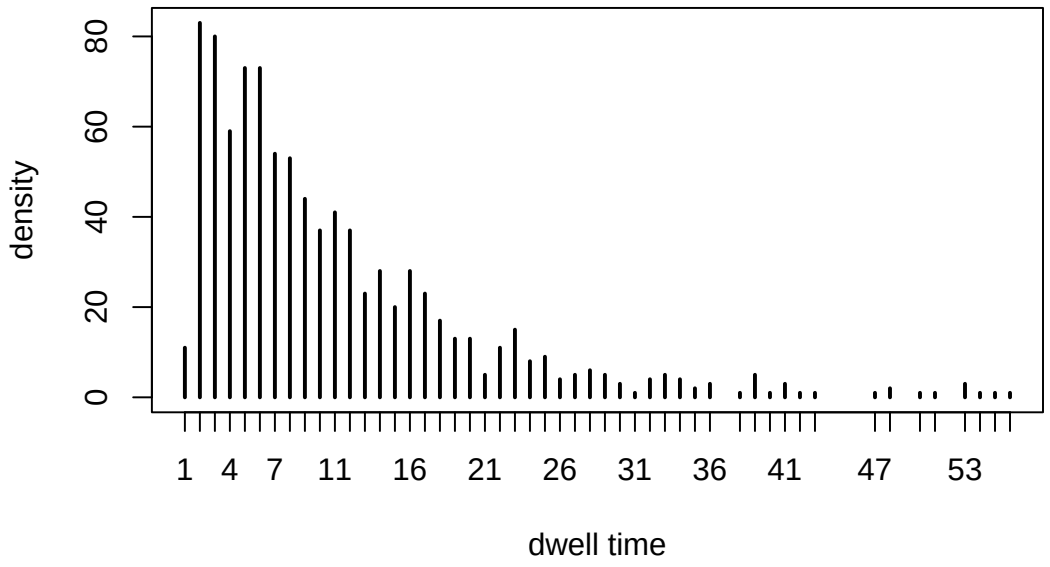
$$\Gamma = \begin{pmatrix} \gamma_{111} & \gamma_{112} & 0 & 0 \\ 0 & 0 & \gamma_{121} & \gamma_{122} \\ \gamma_{211} & \gamma_{212} & 0 & 0 \\ 0 & 0 & \gamma_{221} & \gamma_{222} \end{pmatrix}.$$

Some of the transitions are impossible, and the probabilities are fixed to zero. This can be implemented in hmmTMB using constraints on the transition probabilities.

4.1 Simulate data

We simulate data from a second-order HMM, where the transition probabilities are stored in an array such that γ_{ijk} is on the i-th row, j-th column, and k-th layer. Here, we choose transition probabilities such that (1) after having been in a state for two time steps, there is a 90% probability of staying in that state, and (2) after having switched to a new state, there is a 99% probability of staying in the new state. That is, we have $\gamma_{111} = \gamma_{222} = 0.9$ and $\gamma_{122} = \gamma_{211} = 0.99$, and other transition probabilities can de deduced from these using $\gamma_{ij1} + \gamma_{ij2} = 1$.

```
# State process
tpa <- array(c(0.9, 0.99, 0.01, 0.1, 0.1, 0.01, 0.99, 0.9), dim = c(2, 2, 2))
n <- 1e4
s <- rep(NA, n)
s <- c(1, 2)
for(i in 3:n) {
    s <- c(s, sample(1:2, size = 1, prob = tpa[s[i-2], s[i-1], ]))
}
# Plot distribution of dwell times
plot(table(rle(s)$lengths), xlab = "dwell time", ylab = "density")
```



```r
# Observation process
z <- rnorm(n, mean = c(-5, 5)[s], sd = c(3, 3)[s])
data <- data.frame(z = z)
```

4.2 Second-order HMM

We define the 4-state process $(\tilde{S}_t)$ described above, which requires specifying an initial transition probability matrix with zeros in the right places, and indices of the reference transition probabilities.

```r
tpm0 <- matrix(c(0.9, 0.1, 0, 0,
                 0, 0, 0.1, 0.9,
                 0.9, 0.1, 0, 0,
                 0, 0, 0.1, 0.9),
               ncol = 4, byrow = TRUE)
ref <- c(1, 3, 1, 3)
hid <- MarkovChain$new(data = data, n_states = 4, tpm = tpm0, ref = ref)
```

Note from the definition of $\tilde{S}_t$ that $S_t = 1$ when $\tilde{S}_t = 1$ or 3, and $S_t = 2$ when $\tilde{S}_t = 2$ or 4. So, the observation model is such that parameters in states 1 and 3 are the same, and parameters in states 2 and 4 are the same.

```r
dists <- list(z = "norm")
par0 <- list(z = list(mean = c(-1, 1, -1, 1), sd = c(3, 3, 3, 3)))
obs <- Observation$new(data = data, dists = dists, par = par0)
```

We define parameter constraints to ensure that some transition probabilities are fixed to zero, and that observation parameters have common values in states 1 and 3 and states 2 and 4.

```r
fix_hid <- rep(NA, length(which(is.infinite(hid$coeff_fe()))))
names(fix_hid) <- rownames(hid$coeff_fe())[which(is.infinite(hid$coeff_fe()))]
fix_obs <- c(1, 2, 1, 2, 3, 4, 3, 4)
names(fix_obs) <- rownames(obs$coeff_fe())
fixpar <- list(hid = fix_hid, obs = fix_obs)
```

We create the `HMM` object and fit the model.

```r
hmm <- HMM$new(obs = obs, hid = hid, fixpar = fixpar)
hmm$fit(silent = TRUE)
```

4.3 Results

We can unpack the results to check that the second-order HMM parameters used for simulation were recovered. In this example, the observation parameters and transition probabilities were estimated well. We can also compute the most likely state sequence with the Viterbi algorithm, which shows that the simulated state sequence was recovered.

```
# Observation parameters
obs$par()[,1:2,1]
```

```
       state 1 state 2
z.mean      -5       5
z.sd         3       3
```

```
# Transition probability matrix
hid$tpm()[,,1]
```

```
        state 1 state 2 state 3 state 4
state 1    0.90   0.099   0.000    0.00
state 2    0.00   0.000   0.012    0.99
state 3    0.98   0.016   0.000    0.00
state 4    0.00   0.000   0.103    0.90
```

```
# State sequence
vit <- hmm$viterbi()
vit[which(vit %in% c(1, 3))] <- 1
vit[which(vit %in% c(2, 4))] <- 2
sum(vit == s)/n
```

```
[1] 0.99
```

5 Coupled hidden Markov models

5.1 Cartesian product model

A coupled HMM assumes that two observed variables are driven by two separate, but dependent, state processes (Pohle et al. (2021)). There are several ways to specify them, based on different levels of assumptions about the dependence of the two state processes, and here we focus on the "Cartesian product model" of Pohle et al. (2021).

Similarly to semi-Markov and higher-order models, the basic idea is to rewrite the model as a standard HMM with expanded state space. Consider two observed variables $Z_t^{(1)}$ and

$Z_t^{(2)}$, and assume that they depend on the state processes $S_t^{(1)}$ and $S_t^{(2)}$, respectively. For simplicity, we focus on the 2-state case here, i.e., $S_t^{(1)} \in \{1, 2\}$ and $S_t^{(2)} \in \{1, 2\}$. We define the state variable S_t in such a way that

$$S_t = \begin{cases} 1 & \text{if } S_t^{(1)} = 1 \text{ and } S_t^{(2)} = 1, \\ 2 & \text{if } S_t^{(1)} = 1 \text{ and } S_t^{(2)} = 2, \\ 3 & \text{if } S_t^{(1)} = 2 \text{ and } S_t^{(2)} = 1, \\ 4 & \text{if } S_t^{(1)} = 2 \text{ and } S_t^{(2)} = 2. \end{cases}$$

The Cartesian product model can be written as the 4-state multivariate HMM with state process (S_t).

For this example, we further assume that the observations follow state-dependent normal distributions. The observation model is the following,

$$\begin{cases} \text{State 1:} & Z_t^{(1)} \sim N(\mu_{11}, \sigma_{11}^2), \; Z_t^{(2)} \sim N(\mu_{12}, \sigma_{12}^2) \\ \text{State 2:} & Z_t^{(1)} \sim N(\mu_{11}, \sigma_{11}^2), \; Z_t^{(2)} \sim N(\mu_{22}, \sigma_{22}^2) \\ \text{State 3:} & Z_t^{(1)} \sim N(\mu_{21}, \sigma_{21}^2), \; Z_t^{(2)} \sim N(\mu_{12}, \sigma_{12}^2) \\ \text{State 4:} & Z_t^{(1)} \sim N(\mu_{21}, \sigma_{21}^2), \; Z_t^{(2)} \sim N(\mu_{22}, \sigma_{22}^2) \end{cases}$$

Note that some of the parameters are shared between states. This is because, for example, $Z_t^{(1)}$ has the same distribution in states 1 and 2, because both correspond to $S_t^{(1)} = 1$.

5.2 Simulate data

We simulate from a 4-state Markov chain, where the states are as defined above. We choose transition probabilities such that it is likely for both processes $(S_t^{(1)})$ and $(S_t^{(2)})$ to switch simultaneously. From the 4-state sequence, we can derive the two 2-state sequences for $(S_t^{(1)})$ and $(S_t^{(2)})$.

```
# Transition probabilities for simulation
tpm <- matrix(c(0.9, 0.02, 0.02, 0.06,
                0.02, 0.9, 0.06, 0.02,
                0.02, 0.06, 0.9, 0.02,
                0.06, 0.02, 0.02, 0.9),
              ncol = 4, byrow = TRUE)

# Generate state sequence from 4-state process
n <- 1000
```

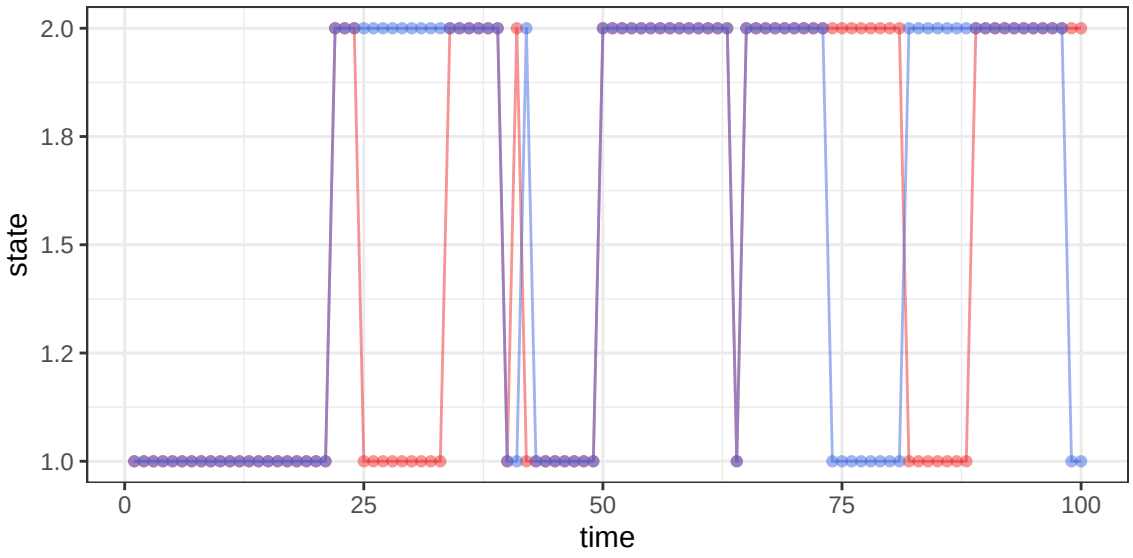
```r
s <- rep(1, n)
for(i in 2:n) {
  s[i] <- sample(1:4, size = 1, prob = tpm[s[i-1],])
}

# Get state sequence for each separate 2-state process
s1 <- ifelse(s %in% c(1, 2), 1, 2)
s2 <- ifelse(s %in% c(1, 3), 1, 2)
```

Both observed variables follow state-dependent normal distributions, and we use the simulated state sequences to generate those.

```r
# State-dependent observation parameters
mean1 <- c(-5, 5)
sd1 <- c(2, 2)
mean2 <- c(0, 10)
sd2 <- c(1, 5)

# Generate observations
z1 <- rnorm(n, mean = mean1[s1], sd = sd1[s1])
z2 <- rnorm(n, mean = mean2[s2], sd = sd2[s2])
data <- data.frame(ID = 1, z1 = z1, z2 = z2, s1 = s1, s2 = s2, time = 1:n)
```

We can look at the simulated state sequences to see that the two processes are dependent but separate: they can take different values, but they tend to switch states simultaneously.

```r
library(ggplot2)
ggplot(data[1:100,], aes(time, s1)) +
  geom_point(col = adjustcolor("firebrick2", 0.5)) +
  geom_line(col = adjustcolor("firebrick2", 0.5)) +
  geom_point(aes(y = s2), col = adjustcolor("royalblue", 0.5)) +
  geom_line(aes(y = s2), col = adjustcolor("royalblue", 0.5)) +
  labs(y = "state")
```



5.3 Model fitting

This model is relatively simple to specify in hmmTMB; the main challenge is to set the correct parameter constraints to ensure that the four states match their definitions. We first create the `MarkovChain` object, which does require any constraints.

```
hid <- MarkovChain$new(data = data, n_states = 4)
```

In the observation model, we want to make sure that the parameters are shared between states as described above (e.g., same mean for $Z_t^{(1)}$ in states 1 and 2, etc.), and we choose the starting parameters accordingly.

```
dists <- list(z1 = "norm", z2 = "norm")
par0 <- list(z1 = list(mean = c(-3, -3, 3, 3), sd = c(1, 1, 1, 1)),
             z2 = list(mean = c(-5, 5, -5, 5), sd = c(2, 3, 2, 3)))

obs <- Observation$new(data = data, dists = dists, par = par0)
```

We set constraints on the observation parameters using the `fixpar` argument in `HMM$new()`, by defining a vector with one integer for each estimated parameter. The value of the integer is used to determine which parameters should be estimated to a common value. To find the list of estimated observation parameters, we can use `obs$coeff_fe()`; this is also handy to set the names of the vector of constraints.

```
fixpar_obs <- c(1, 1, 2, 2, 3, 3, 4, 4, 5, 6, 5, 6, 7, 8, 7, 8)
names(fixpar_obs) <- rownames(obs$coeff_fe())
```

```
fixpar <- list(obs = fixpar_obs)
```

We can now create the HMM and fit it. The estimated parameters closely match the parameters used to simulate the data.

```
hmm <- HMM$new(obs = obs, hid = hid, fixpar = fixpar)
hmm$fit(silent = TRUE)
hmm$par()
```

```
$obspar
, , 1

        state 1 state 2 state 3 state 4
z1.mean   -5.05      -5    5.03       5
z1.sd      2.00       2    2.01       2
z2.mean   -0.02      10   -0.02      10
z2.sd      1.00       5    1.00       5


$tpm
, , 1

        state 1 state 2 state 3 state 4
state 1   0.878   0.022   0.033   0.067
state 2   0.021   0.901   0.064   0.013
state 3   0.010   0.099   0.876   0.015
state 4   0.032   0.032   0.031   0.905
```

References

Langrock, Roland, and W Zucchini. 2011. "Hidden Markov Models with Arbitrary State Dwell-Time Distributions." *Computational Statistics & Data Analysis* 55 (1): 715–24.

Pohle, Jennifer, Roland Langrock, Mihaela van der Schaar, Ruth King, and Frants Havmand Jensen. 2021. "A Primer on Coupled State-Switching Models for Multiple Interacting Time Series." *Statistical Modelling* 21 (3): 264–85.

Zucchini, Walter, Iain L MacDonald, and Roland Langrock. 2017. *Hidden Markov Models for Time Series: An Introduction Using R, Second Edition*. CRC Press.