hmmTMB technical guide

Théo Michelot, Richard Glennie

2020-11-16

1 Introduction

The package hmmTMB implements hidden Markov models with flexible formulations for the parameters of the hidden state process and parameters of the observation distributions, including linear, smooth or random effects of covariates. This document provides some technical background on the implementation, and is only intended to users who would like to get an idea of the inner workings of the package. For a more accessible introduction to the package, including example analyses with detailed code, see the other vignette ('hmmTMB user guide').

2 Mathematical background

2.1 Hidden Markov models

A hidden Markov model (HMM) is comprised of two random processes:

- the state process (S_t), defined as a J-state Markov chain, such that $S_t \; in \{1, 2, \dots, J\}$ at any time $t = 1, 2, \dots$;
- the (possibly multivariate) observation process (Z_t). At each time t, the observation Z_t arises from one of J probability distributions, determined by the value S_t of the state process.

The state process is parameterised by an initial distribution $\pi^{(1)} = [\Pr(S_1 = 1), \dots, \Pr(S_1 = J)]$, and by a transition probability matrix

$$\Gamma = \begin{pmatrix} \gamma_{11} & \gamma_{12} & \cdots & \gamma_{1J} \\ \gamma_{21} & \gamma_{22} & \cdots & \gamma_{2J} \\ \vdots & \vdots & \ddots & \vdots \\ \gamma_{J1} & \gamma_{J2} & \cdots & \gamma_{JJ} \end{pmatrix},$$

where the (i, j)-th entry is the probability of a transition from state i to state j, i.e.,

$$\gamma_{ij} = \Pr(S_t = j | S_t = i).$$

The observations are typically assumed to arise from some parametric distribution, with one set of parameters for each possible value of the state process, and we can write

$$p(Z_t = z_t | S_t = j) = f_Z(z_t, \theta_j)$$

where f_Z is the assumed pdf for the observation, and θ_j is the vector of parameters for state j.

The main goal of hmmTMB is to offer the possibility to specify flexible models for the transition probabilities or the state-dependent observation parameters, including fixed and random effects of covariates, and smooth relationships between parameters and covariates using smoothing splines. Here, we only provide a very succinct introduction to HMMs to explain how this is done in the package; for a comprehensive description, see for example Zucchini, MacDonald, and Langrock (2016).

2.2 Basis-penaly smoothing splines

Flexible relationships between the model parameters described in the previous section and covariates can be defined using basis-penalty smoothing splines, similarly to generalized additive models (GAMs; Wood (2017)). In this framework, a parameter θ (either a transition probability or a parameter of the observation distribution) can be specified at time t by

$$h(\theta_t) = \beta_0 + f_1(x_{1t}) + f_2(x_{2t}) + \ldots,$$

where h is a link function, β_0 is an intercept parameter, and the functions $f_1, f_2, \ldots$ define the smooth relationships between θ_t and the covariates $x_{1t}, x_{2t}, \ldots$. The smooth functions are written as linear combinations of basis functions,

$$f_i(x) = \sum_{k=1}^{K} \beta_{ik}\psi_{ik}(x),$$

where various bases can be chosen (e.g., cubic splines, B-splines). Following standard GAM methodology, the roughness ('wiggliness') of these functions is then penalised in the likelihood of the model to ensure that they are smooth.

Consider that we have n observations $z_1, \ldots, z_n$ from this model. The penalised log-likelihood is

$$l_p(\alpha, \beta, \lambda | z_1, \ldots, z_n) = \log\{L(\alpha, \beta | z_1, \ldots, z_n)\} - \sum_i \lambda_i \beta_i^T S_i \beta_i,$$

where $L(\alpha, \beta | z_1, \ldots, z_n)$ is the unpenalised HMM likelihood of parameters α and β (e.g. computed using the forward algorithm), λ is a vector of smoothness parameters, and S_i is a known penalty matrix determined by the choice of basis.

The smoothness parameters λ_i are not known, and need to be estimated jointly with other model parameters. In hmmTMB, we use the marginal likelihood method; i.e., we treat the basis coefficients as random effects, and we consider the marginal likelihood of the fixed effects α and smoothness parameters λ,

$$\begin{aligned} L(\alpha, \lambda | z_1, \ldots, z_n) &= \int \exp\{l_p(\alpha, \beta, \lambda | z_1, \ldots, z_n)\} d\beta \\ &= \int [z_1, \ldots, z_n | \alpha, \beta][\beta | \lambda] d\beta \end{aligned}$$

where $[z_1, \ldots, z_n | \alpha, \beta] = L(\alpha, \beta | z_1, \ldots, z_n)$ is the HMM likelihood, and $[\beta|\lambda]$ is a multivariate normal pdf with mean zero and block-diagonal precision matrix with blocks $\lambda_i S_i$. TMB uses the Laplace approximation on the integrand to evaluate this expression. We can then perform maximum likelihood estimation on the marginal likelihood to obtain estimates of α and λ.

3 Implementation using TMB and mgcv

3.1 Model specification using mgcv

We use the gam function from the package mgcv to create design matrices from the formulas specified for the HMM parameters (including linear effects, basis functions, and random effects), and the penalty matrices for smoothing splines (Wood (2017)). Users therefore need to use the mgcv syntax for smooth terms and random effects, as described in the mgcv documentation. Here, we present a short example to illustrate how gam is used in hmmTMB. Consider the following data frame containing values for two covariates x1 and x2}, and a time series identifier ID,

```
  ID          x1          x2
1  1 -0.77013961 -0.71633943
2  1 -1.68012102 -0.49090457
3  1 -0.40506564  0.02080336
```

```
4  1 -0.57909035  0.09380136
5  1 -0.06745051 -1.44970979
6  1  0.06613943 -0.79080510
```

Then, consider that we want to specify one of the HMM parameters with the following formula:

```
form <- ~ x1 + s(x2, k = 5, bs = "cc") + s(ID, bs = "re")
```

where x1 has a linear effect, x2 has a smooth effect modelled using cyclic cubic regression splines, and a random normal intercept is included for ID. We compute the model matrices as follows,

```
# Create smooth object using mgcv
smooth <- gam(formula = update(form, dummy ~ .),
              data = cbind(dummy = 1, data),
              fit = FALSE)

# Design matrix
X <- smooth$X

# Number of non-smooth model terms (i.e., fixed effects)
nsdf <- smooth$nsdf

# Design matrix for fixed effects
X_fe <- X[, 1:nsdf, drop = FALSE]

# Design matrix for random effects (including smooth model terms)
X_re <- X[, -(1:nsdf), drop = FALSE]
```

The design matrix for the fixed effects is

```
head(X_fe)
```

```
  (Intercept)          x1
1           1 -0.77013961
2           1 -1.68012102
3           1 -0.40506564
4           1 -0.57909035
5           1 -0.06745051
6           1  0.06613943
```

and the design matrix for the random effects is

```
head(X_re)
```

```
                                  ID1 ID2 ID3 ID4
1  0.4711597  0.2504715 -0.25556677   1   0   0   0
2  0.2330795  0.5487361 -0.24766033   1   0   0   0
3 -0.2195417  0.9256027 -0.10925643   1   0   0   0
4 -0.2549379  0.9214833 -0.06793577   1   0   0   0
5  0.7882939 -0.4770955 -0.33498326   1   0   0   0
6  0.5430424  0.1494460 -0.25639169   1   0   0   0
```

where the first three columns correspond to the four basis functions for x2, and the four last columns are dummy indicator variables for ID. Then, the linear predictor for the corresponding HMM parameter is

```
lp <- X_fe %*% coeff_fe + X_re %*% coeff_re
```

where coeff_fe and coeff_re are the coefficients for the fixed effects and the random effects, respectively. The HMM parameter (for each data row) is then obtained by applying the inverse link function to this linear

predictor.

Similarly, the penalty matrix for the smooth terms can be extracted from the GAM object,

```
S <- smooth$S
```

3.2 Model fitting using TMB

We use the R package Template Model Builder (TMB) to implement the marginal likelihood of the model, based on the Laplace approximation to integrate over the random effects (Kristensen et al. (2016)). Here, we present the general idea for using TMB to evaluate the negative log-likelihood, and to obtain point and uncertainty estimates for the model parameters.

The penalised log-likelihood of the fixed and random effects must first be written in C++, following the TMB syntax (for examples, see the TMB documentation). For our model, it has two main components:

- the forward algorithm is used to evaluate the HMM log-likelihood given all parameters, $\log\{L(\alpha, \beta|z_1, \ldots, z_n)\}$. This part requires the HMM parameters on a time grid, which can be computed based on the model matrices provided by mgcv, as described in the previous section.
- the log-pdf of the basis coefficients (and other random effects) given the smoothness parameter, $\log[\beta|\lambda]$, obtained as the log of a multivariate normal pdf with block-diagonal precision matrix, where the i-th block is $\lambda_i S_i$. The smoothness matrix S_i is provided by mgcv, as described in the previous section.

The negative penalised log-likelihood function (and its gradient) can then be defined in R with the TMB function MakeADFun, using the argument 'random' to specify that the basis coefficients coeff_re should be treated as random effects. We then use the numerical optimiser optim to perform maximum likelihood estimation of the fixed effect parameters α and λ. After optimisation, we use the function sdreport to obtain estimates of the random effect parameters β, as well as a joint precision matrix for the fixed and random effects. Posterior samples of all model parameters can be generated from a multivariate normal distribution, where the covariance matrix is the inverse of this precision matrix.

Note that TMB relies on the Laplace approximation to integrate the likelihood over the random effects, i.e. the likelihood is approximated by a normal pdf to make the integration tractable. This may result in large errors in cases where the likelihood is very non-normal, e.g., if it is multimodal. HMM likelihoods are known to be multimodal, in particular along the parameters of the observation distributions, and it is not clear how this will affect estimation. Recently, McClintock (2020) found that TMB performed well in various simulation scenarios for HMMs with random effects, but further investigations will be required to understand this issue better.

4 Package structure

We use the object-oriented programming framework of the package R6 (Chang (2019)) for all data and model objects. The main classes are shown in the table below.

Class name	What it is	Main attributes	Example
Dist	Probability distribution	name	normal distribution
		pdf	dnorm
		rng	rnorm
		link functions	
		inverse link functions	
MarkovChain	Markov chain model	number of states	2-state Markov chain
		formulas	no covariate dependence
		Markov chain parameters	transition probabilities
Observation	Observation model	data frame	data frame of observations (z1, z2)

Class name	What it is	Main attributes	Example
		list of Dist objects formulas observation parameters	list(z1 = "norm", z2 = "gamma") no covariate dependence parameters of normal and gamma distributions
HMM	Hidden Markov model	Markov chain object Observation object output of optimiser	

Each class is defined in a separate R file, which contains all its methods (i.e., the functions that can be applied to an object of that class).

4.1 Observation distribution

Distributions for the observation process are created using the Dist class. To define a new distribution, we provide its name (as a string), its probability density/mass function (as a function), the random generator function, and the link functions for its parameters (as named lists of functions). For example, for the normal distribution, we have

```
dist_norm <- Dist$new(
  name = "norm",
  pdf = dnorm,
  rng = rnorm,
  link = list(mean = identity, sd = log),
  invlink = list(mean = identity, sd = exp),
  npar = 2
)
```

The Poisson, gamma, normal, beta, and von Mises distributions are included in the package, and more will be implemented in the future.

The functions n2w and w2n transform parameters of the observation distributions from the natural to the working scale and vice-versa. We can verify that applying the two functions successively returns the original parameter values:

```
# List of parameters (on the natural scale)
par1 <- list(mean = c(0, 5), sd = c(1, 10))

# Transform to working scale
wpar <- dist_norm$n2w(par1)
wpar
```

```
   mean1    mean2      sd1      sd2
0.000000 5.000000 0.000000 2.302585
```

```
# Transform back to natural scale
par2 <- dist_norm$w2n(wpar)
par2
```

```
$mean
mean1 mean2
    0     5

$sd
sd1 sd2
```

```
  1  10
```

4.2 HMM observation model

The class Observation encapsulates the model for the observed variables.

```
# Create a dummy data set
times <- seq(as.POSIXct("2020/01/01 00:00:00"), by = "hour", length = 100)
data <- data.frame(ID = rep(1, 100),
                   y = rnorm(100),
                   time = times,
                   x = cumsum(runif(100)))

# List of observation distributions
dists <- list(y = "norm")

# List of observation parameters
par <- list(y = list(mean = c(0, 0), sd = c(1, 10)))

# Number of states
n_states <- 2

# Create observation process object
obs_process <- Observation$new(data = data, dists = dists,
                               n_states = n_states, par = par)
```

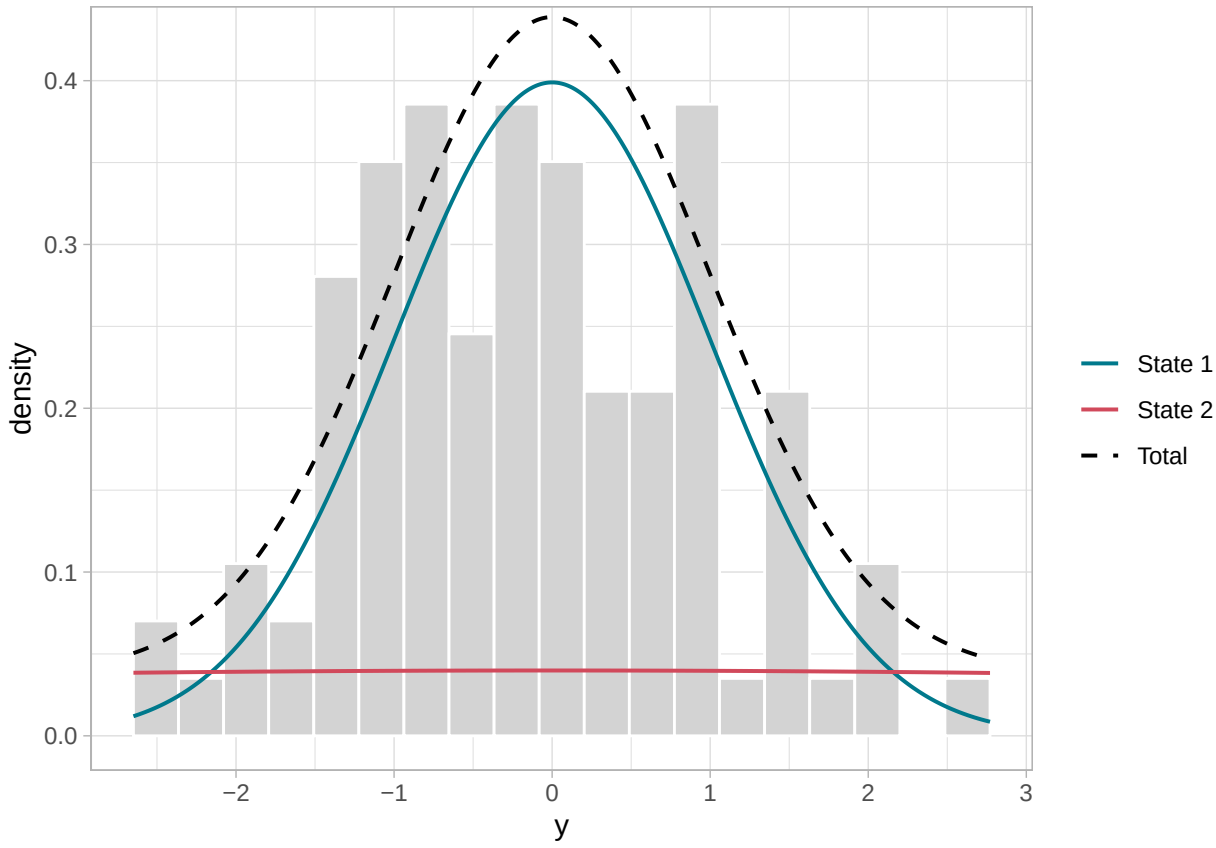
The parameters on the working scale are calculated when the object is created:

```
obs_process$coeff_fe()
```

```
                              [,1]
y.mean.state1.(Intercept) 0.000000
y.mean.state2.(Intercept) 0.000000
y.sd.state1.(Intercept)   0.000000
y.sd.state2.(Intercept)   2.302585
```

The function plot_dist generates a histogram of the observations with the probability density (or mass) function of the specified distribution.

```
# Plot histogram and pdf for variable "x"
obs_process$plot_dist(name = "y")
```



4.3 Hidden state process model

The state process model is defined with the MarkovChain class, which includes the number of states and covariate formulas for the transition probabilities. In the following example, we create a 2-state Markov chain with a linear relationship between γ_{12} and the covariate x, and a smooth relationships between γ_{21} and x (using a cubic spline with 5 basis functions).

```
structure <- matrix(c(".", "~x",
                      "~s(x, k=5, bs=\"cr\")", "."),
                    nrow = 2, byrow = TRUE)
hid_process <- MarkovChain$new(n_states = 2, structure = structure, data = data)
```

If they are not specified, the parameters of the Markov Chain are initialised such that all effects are zero and the diagonal elements of the transition probability matrix are 0.9.

```
# Fixed effect parameters
hid_process$coeff_fe()
```

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

```
# Random effect parameters
hid_process$coeff_re()
```

```
            [,1]
S2>S1.s(x).1    0
S2>S1.s(x).2    0
```

```
S2>S1.s(x).3    0
S2>S1.s(x).4    0
```

We can simulate from this Markov chain model using the simulate method.

```
# Number of time steps to simulate
n_sim <- 10

# New data frame of covariates, to use in simulation
new_data <- data.frame(ID = 1,
                       x = cumsum(runif(n_sim)))

hid_process$simulate(n = n_sim, data = data, new_data = new_data)
```

```
Simulating states... 20%Simulating states... 30%Simulating states... 40%Simulating states... 50%Simulati
```

```
 [1] 2 2 2 2 2 2 2 2 2 2
```

4.4 Hidden Markov model class

The HMM class encapsulates a MarkovChain and an Observation object, the two components of an HMM. It include methods to fit the model and visualise the results.

```
hmm <- HMM$new(obs = obs_process, hidden = hid_process)
```

5 Some implementation details

5.1 Creation of model matrices (make_matrices)

The function make_matrices, used inside the classes MarkovChain and Observation, creates design matrices (of fixed and random effects) and smoothing matrices (smooths and random effects) from model formulas. Formulas must be specified using the mgcv syntax, and the heavy lifting is done by the function gam.

More specifically, when a formula and a data frame are passed as input, the function gam can return the design matrix X and the number of columns of fixed effects (which we use to split X into a design matrix of fixed effects and a design matrix of random effects). It also returns the smoothing matrix S if there are smooth terms or random effects. The function make_matrices loops over a list of formulas (e.g., formulas for MarkovChain model), and creates three block-diagonal matrices: the design matrix of fixed effects X_fe, the design matrix of random effects X_re, and the smoothing matrix S.

5.2 Model parameters

The parameters for each model component (i.e., hidden state process and observation process) are stored in three vectors:

- coeff_fe: Vector of parameters for the fixed effects. If there are no covariates, this is just the HMM parameters transformed by the link functions (i.e., intercepts of linear predictor). If there are fixed effects of covariates, this is a vector of regression coefficients.
- coeff_re: Vector of parameters for the random effects. If there are no smooth/random effects, this is empty. If there are smooth effects, these are the coefficients for the corresponding basis functions.
- lambda: Vector of smoothness parameters. If there are no smooth/random effects, this is empty. If there are smooth effects, this coefficient measures the smoothness of the corresponding spline. If there are iid normal random effect, lambda can be linked to the variance (see vcomp method).

5.3 Uncertainty estimates using CI and predict functions

TMB returns standard errors for the estimated fixed effect parameters, and the method HMM$CI_coeff uses these to derive confidence intervals. Obtaining confidence intervals for the HMM parameters (transition probabilities and parameters of the observation distributions), which is typically what we want, is a little harder because those parameters can depend on covariates.

We create confidence intervals for the HMM parameters using simulations. The function sdreport in TMB can return a joint precision matrix for all model parameters if the option getJointPrecision = TRUE is specified. Based on the asymptotic normality of the maximum likelihood estimates (MLEs), we generate a large number of parameter sets from a multivariate normal distribution with mean the MLEs and covariance matrix given by the inverse of the precision matrix provided by TMB. For each simulated set of parameters, we derive the corresponding HMM parameters, and we obtain approximate pointwise confidence intervals by extracting the appropriate quantiles.

6 Future work

6.1 Higher priority

- additional observation distributions
  - Weibull
  - log-normal
  - wrapped Cauchy
  - Bernoulli...

6.2 Lower priority (easy?)

- state probabilities (local decoding)
- pseudo-residuals
- allow for input of known states (semi-supervised learning)
- stationary = TRUE/FALSE option in MarkovChain class

6.3 Lower priority (difficult?)

- AIC
- zero inflation
- improve parametrisation of mean of circular distributions (avoid numerical issues around -pi and pi)
- allow for constraints on parameters
  - inequalities, e.g. $\mu_1 < \mu_2$
  - bounds, e.g. $-1 < \mu_1 < 1$ (using custom link function)
  - fixed value, e.g. $\mu_1 = 0$ (using map)
- obtain starting values for complex models from simpler nested model (e.g. for smooth effects)
- method "update" similar to glmmTMB, to create a new model from an existing model?

References

Chang, W. 2019. *R6: Encapsulated Classes with Reference Semantics.* https://CRAN.R-project.org/package=R6.

Kristensen, Kasper, Anders Nielsen, Casper Willestofte Berg, Hans J Skaug, and Brad Bell. 2016. "TMB: Automatic Differentiation and Laplace Approximation." *Journal of Statistical Software* 70 (5): 1–21.

McClintock, Brett T. 2020. "Worth the Effort? A Practical Examination of Random Effects in Hidden Markov Models for Animal Telemetry Data." *BioRxiv*.

Wood, S. N. 2017. *Generalized Additive Models: An Introduction with R*. 2nd ed. Chapman; Hall/CRC.

Zucchini, Walter, Iain L MacDonald, and Roland Langrock. 2016. *Hidden Markov Models for Time Series: An Introduction Using R, Second Edition*. CRC press.