

Fast parameter and confidence interval estimation for Hidden Markov Models using Template Model Builder

Timothee Bacri^{*1}, Jan Bulla¹, and Geir D. Berentsen²

¹ Department of Statistics, University of Bergen, 5007 Bergen, Norway

² Department of Business and Management Science, Norwegian School of Economics, Helleveien 30, 5045 Bergen, Norway

Received zzz, revised zzz, accepted zzz

Hidden Markov Models (HMMs) are a class of models widely used in speech recognition (See e.g. Gales and Young, 2008; Fredkin and Rice, 1992). There are straightforward ways to compute Maximum Likelihood (ML) estimates of their parameters, but getting confidence intervals can be more difficult (See e.g. Zucchini et al., 2016; Lystig and Hughes, 2002). In addition, computing ML estimates can be time-consuming when the dataset and the complexity become very large. We show in this paper a way to speed up computation by up to 50 times in the language R when compared to usual optimization approaches, and at the same time retrieve standard errors easily. In a first part, we see how to optimize HMMs with the TMB package in R and how to retrieve confidence intervals. In a second part, we compare different optimizers such as `nlminb`, and minimize the negative log-likelihood directly on different datasets: a small one (240 data points) from Leroux and Puterman (1992), a medium sized simulated one (2000 data points), and a large one (87648 data points) from a hospital.

Key words: Hidden Markov Model; TMB; Confidence intervals;

Supporting Information for this article is available from the author or on the WWW under <https://github.com/timothee-bacri/hmm-tmb>

1 Introduction

TODO: -define max. l. estimation (MLE)

Hidden Markov models (HMMs) are a versatile type of model that have been employed in many different situations since their introduction by Baum and Petrie (1966). As an example, Fredkin and Rice (1992) applied them in speech recognition, Lystig and Hughes (2002) to rainfall occurrence data, and Schadt et al. (1998) to phylogenetic trees. Fredkin and Rice (1992) and Zucchini et al. (2016) are references in the theory of HMMs. Evaluating uncertainty and getting confidence intervals in HMMs is uneasy and not necessarily reliable. Although Cappé et al. (2006, Ch. 12) showed it can be achieved using asymptotic normality of the ML estimates of the parameters under certain conditions, Frühwirth-Schnatter (2006, p. 53) points out that in independent mixture models, “The regularity conditions are often violated”. McLachlan and Peel (2004, p. 68) adds that “In particular for mixture models, it is well known that the sample size n has to be very large before the asymptotic theory of maximum likelihood applies.” Lystig and Hughes (2002) shows a way to compute the exact Hessian, and Zucchini et al. (2016) shows another way to compute the approximate Hessian and thus confidence intervals but admits that “the use of the Hessian to compute standard errors (and thence confidence intervals) is unreliable if some of the parameters are on or near the boundary of their parameter space”.

^{*}Corresponding author: e-mail: timothee.bacri@uib.no, Phone: +33-636-775-063

TMB (Template Model Builder) is an R package for efficient fitting of complex statistical random effect models to data, as described by Kristensen *et al.* (2015). It provides exact calculations of first and second order derivatives of the likelihood of a model by automatic differentiation, which allows for efficient gradient and/or Hessian based optimization of the likelihood as well as uncertainty estimates by means of the Hessian. The Hessian is not necessarily directly applicable for evaluating parameter uncertainty in HMMs as there are several parameter constraints in these models. This can be addressed by constraint optimization, and subsequently combining the Hessian with the Jacobian of the constraints to obtain the covariance matrix as shown by Visser *et al.* (2000). Alternatively, Zucchini *et al.* (2016) shows how the constraints can be imposed by suitable transformations of the parameters. The covariance matrix of the untransformed (original) parameters can then be retrieved by the delta method, a feature implemented in TMB.

In this paper, we first show how to optimize Poisson HMMs with the help of TMB in the language R. Afterwards, we explain how to make a nested model through an example, how to effortlessly compute confidence intervals, how to fetch interesting probabilities, and we apply a Poisson HMM on a real hospital dataset. Eventually, we see that TMB can accelerate traditional optimizers in R by up to approximately 50 times on a fairly large dataset, and can easily output confidence intervals similar to bootstrap and profile methods via its Hessian based approach.

Maximum likelihood estimation can be achieved either by a direct numerical maximization as introduced by Turner (2008) and later detailed with R code by Zucchini *et al.* (2016), by an Expectation-Maximization (EM) based algorithm as described by Baum *et al.* (1970), or by a mixture of the 2 as shown by Bulla and Berzel (2008). We decide to use a direct maximization approach instead of the EM algorithm. The reason is that the direct maximization approach is easier to adapt if one wants to fit different and more complex models. It also deals easily with missing observations, whereas the EM approach is more complex.

2 Principles of using TMB for Maximum Likelihood Estimation

In order to keep this tutorial at acceptable length, all sections follow the same concept. That is, the reader is encouraged to consult our GitHub repository in parallel ([https://github.com/\(TIMO:MAKE A REPOSITORY\)](https://github.com/(TIMO:MAKE A REPOSITORY))). This permits to copy-paste or download all the scripts presented in this tutorial for each section. Moreover, the repository also contains additional explanations, comments, and scripts.

2.1 Setup

Execution of our routines requires the installation of the R-package TMB and the software Rtools, where the latter serves for compiling C++ code. In order to ensure reproducibility of all results involving the generation of random numbers, the `set.seed` function requires R version number 3.6.0 or greater. Our scripts were tested on an Intel(R) Core(TM) i7-8700 processor running under Windows 10 Enterprise version 1809.

In particular for beginners, those parts of scripts involving C++ code can be difficult to debug because the code operates using a specific template. Therefore it is helpful to know that TMB provides a debugging feature, which can be useful to retrieve diagnostic error messages, in RStudio. Enabling this feature is optional and can be achieved by the command `TMB:::setupRStudio()` (requires manual confirmation and re-starting RStudio).

2.2 Linear regression example

We begin by demonstrating the principles of TMB, which we illustrate through the fitting procedure for a simple linear model. This permits, among other things, to show how to handle parameters subject to constraints, an aspect particularly relevant for HMMs. A more comprehensive tutorial on TMB presenting

many technical details in more depths is available at https://kaskr.github.io/adcomp/_book/Tutorial.html.

Let \mathbf{x} and \mathbf{y} denote the predictor and response vector, respectively, both of length n . For a simple linear regression model with intercept a and slope b , the negative log-likelihood equals

$$-\log L(a, b, \sigma^2) = -\sum_{i=1}^n \log(\phi(y_i; a + bx_i, \sigma^2)),$$

where $\phi(\cdot; \mu, \sigma^2)$ corresponds to the density function of the univariate normal distribution with mean μ and variance σ^2 .

The use of TMB requires the (negative) log-likelihood function to be coded in C++ under a specific template, which is then loaded into R. The minimization of this function and other post-processing procedures are all carried out in R. Therefore, we require two files. The first file, named *linreg.cpp*, is written in C++ and defines the objective function, i.e. the negative log-likelihood (nll) function of the linear model, as follows.

```
#include <TMB.hpp> //import the TMB template

template<class Type>
Type objective_function<Type>::operator() ()
{
  DATA_VECTOR(y); // Data vector y passed from R
  DATA_VECTOR(x); // Data vector x passed from R

  PARAMETER(a); // Parameter a passed from R
  PARAMETER(b); // Parameter b passed from R
  PARAMETER(tsigma); // Parameter sigma (transformed, on log-scale)
                      // passed from R

  // Transform tsigma back to natural scale
  Type sigma = exp(tsigma);

  // Declare negative log-likelihood
  Type nll = - sum(dnorm(y,
                        a + b * x,
                        sigma,
                        true));

  // Necessary for inference on sigma, not only tsigma
  ADREPORT(sigma);

  return nll;
}
```

Note that we define data inputs x and y using the `DATA_VECTOR()` declaration in the above code. Furthermore, we declare the nll as a function of the three parameters a , b and $\log(\sigma)$ using the `PARAMETER()` declaration. In order to be able to carry out unconstrained optimization procedures in the following, the nll function is parameterized in terms of $\log(\sigma)$. While the parameter σ is constrained to be non-negative,

$\log(\sigma)$ can be freely estimated. Alternatively, constraint optimization methods could be carried out, but we do not investigate such procedures. The `ADREPORT()` function is optional but useful for parameter inference at the postprocessing stage.

The second file needed is written in R and serves for compiling the nll function defined above and carrying out the estimation procedure by numerical optimization of the nll function. The .R file (shown below) carries out the compilation of the C++ file and minimization of the nll function:

```
# Loading TMB package
library(TMB)
# Compilation. The compiler returns 0 if the compilation of
# the cpp file was successful
TMB::compile("code/linreg.cpp")

## [1] 0

# Dynamic loading of the compiled cpp file
dyn.load(dynlib("code/linreg"))
# Generate the data for our test sample
set.seed(123)
data <- list(y = rnorm(20) + 1:20, x = 1:20)
parameters <- list(a = 0, b = 0, tsigma = 0)
# Instruct TMB to create the likelihood function
obj_linreg <- MakeADFun(data, parameters, DLL = "linreg",
                        silent = TRUE)
# Optimization of the objective function with nlminb
mod_linreg <- nlminb(obj_linreg$par, obj_linreg$fn)
mod_linreg$par

##           a           b          tsigma
## 0.31009240 0.98395535 -0.05814659
```

In addition to the core functionality presented above, different types of post-processing of the results are possible as well. For example, the function `sdreport` returns the ML estimates and standard errors of the parameters in terms of which the nll is parameterized:

```
sdreport(obj_linreg, par.fixed = mod_linreg$par)

## sdreport(.) result
##           Estimate Std. Error
## a           0.31009240 0.43829083
## b           0.98395535 0.03658781
## tsigma -0.05814659 0.15811381
## Maximum gradient component: 6.931683e-05
```

In principle, the argument `par.fixed = mod_linreg$par` is optional but recommended, because it ensures that the `sdreport` function is carried out at the minimum found by `nlminb`. Note that the standard errors above are based on the Hessian matrix of the nll.

From a practical perspective, it is usually desirable to obtain standard errors for the constrained variables, in this case σ . To achieve this, one can run the `summary` function with argument `select = "report"`:

```
summary(sdreport(obj_linreg, par.fixed = mod_linreg$par),
        select = "report")

##           Estimate Std. Error
## sigma 0.9435116   0.1491822
```

These standard errors result from the generalized delta method described by Kass and Steffey (1989), which is implemented within TMB. Note that full functionality of the `sdreport` function requires calling the function `ADREPORT` on the additional parameters of interest (i.e. those including transformed parameters, in our example σ) in the C++ part. The `select` argument restricts the output to variables passed by `ADREPORT`. This feature is particularly useful when the likelihood has been reparameterized as above, and is especially relevant for HMMs. Following Zucchini et al. (2016), we refer to the original parameters as natural parameters, and to their transformed version as the working parameters. Last, we display the estimation results from the `lm` function for comparison.

```
lm(y ~ x, data = data)$coefficients

## (Intercept)          x
##    0.3100925    0.9839554
```

Note that minor deviations from the results of `lm` originate in the numerical methods involved in the selected optimization procedure, in our case `nlminb`.

3 Parameter estimation techniques for HMMs

In this section we recall basic concepts underlying parameter estimation for HMMs via direct numerical optimization of the likelihood. In terms of notation, we stay as close as possible to Zucchini et al. (2016), where a more detailed presentation is available.

3.1 Basic notation and model setup

A large variety of modeling approaches is possible with HMMs, ranging from rather simple to highly complex setups. In a basic HMM, one assumes that the data-generating process corresponds to a time-dependent mixture of the so-called conditional distributions. More specifically, the mixing process is driven by an unobserved (hidden) homogeneous Markov chain. In this paper we focus on a Poisson HMM, but only small changes are necessary to adapt our scripts to models with other conditional distributions. Let $\{X_t : t = 1, \dots, T\}$ and $\{C_t : t = 1, \dots, T\}$ denote the observed and hidden process, respectively. For an m -state Poisson HMM, the conditional distributions with parameter λ_i are then specified through

$$p_i(x) = P(X_t = x | C_t = i) = \frac{e^{-\lambda_i} \lambda_i^x}{x!},$$

where $i = 1, \dots, m$. Furthermore, we let $\mathbf{\Gamma} = \{\gamma_{ij}\}$ and $\boldsymbol{\delta}$ denote the transition probability matrix (TPM) of the Markov chain and the corresponding stationary distribution, respectively. It is noteworthy that Markov chains in the context of HMMs are often assumed irreducible and aperiodic. For example, Grimmett et al. (2001, Lemma 6.3.5 on p. 225 and Theorem 6.4.3 on p. 227) show that irreducibility ensures the existence of the stationary distribution, and Feller (1968, p. 394) describe that aperiodicity implies that a unique limiting distribution exists and corresponds to the stationary distribution. These results are, however, of limited relevance for most estimation algorithms, because the elements of $\mathbf{\Gamma}$ are in general strictly positive. Nevertheless, one should be careful when manually setting selected elements of $\mathbf{\Gamma}$ equal to zero.

3.2 The likelihood function of an HMM

The likelihood function of an HMM requires, in principle, an summation over all possible state sequences. As shown e.g. by Zucchini et al. (2016, p. 37), a computationally convenient representation as a product of matrices is possible. Let $X^{(t)} = \{X_1, \dots, X_t\}$ and $x^{(t)} = \{x_1, \dots, x_t\}$ denote the history of the observed process X_t and the observations x_t from time zero up to time t . Moreover, let θ denote the vector of model parameters, which consists of the parameters of the TPM and the parameters of the conditional probability density functions (pdf). Given these parameters, the likelihood of the observations $\{x_1, \dots, x_T\}$ can then be expressed as

$$L(\theta) = \mathbf{P}(X^{(T)} = x^{(T)}) = \delta \mathbf{P}(x_1) \mathbf{\Gamma P}(x_2) \mathbf{\Gamma P}(x_3) \dots \mathbf{\Gamma P}(x_T) \mathbf{1}', \quad (1)$$

where

$$\mathbf{P}(x) = \begin{pmatrix} p_1(x) & & & 0 \\ & p_2(x) & & \\ & & \ddots & \\ 0 & & & p_m(x) \end{pmatrix}$$

corresponds to a diagonal matrix with the m conditional pdfs evaluated at x (we will use the term density despite the discrete support), and $\mathbf{1}$ denotes a vector of ones. The first element of the likelihood function, the so-called initial distribution, is given by the stationary distribution δ here. Alternatively, the initial distribution may be estimated freely, which requires minor changes to the likelihood function discussed in Section 4.1.

Note that the treatment of missing data is comparably straightforward in this setup. If x is a missing observations, one just has to set $p_i(x) = 1$, thus $\mathbf{P}(x)$ reduces to the unity matrix as detailed in Zucchini et al. (2016, p. 40). Zucchini et al. (2016, p. 41) also explains how to adjust the likelihood when entire intervals are missing. Furthermore, this representation of the likelihood is quite natural from an intuitive point of view. From left to right, it can be interpreted as a pass through the observations: one starts with the initial distribution multiplied by the conditional density of x_1 collected in $\mathbf{P}(x_1)$. This is followed by iterative multiplications with the TPM modeling the transition to the next observation, and yet another multiplication with contributions of the following conditional densities.

3.3 Forward algorithm and backward algorithm

The pass through the observations described above actually forms the basis for an efficient evaluation of the likelihood function. More precisely, the so-called “forward algorithm” allows for a recursive computation of the likelihood. For setting up this algorithm, we need to define the vector α_t by

$$\begin{aligned} \alpha_t &= \delta \mathbf{P}(x_1) \mathbf{\Gamma P}(x_2) \mathbf{\Gamma P}(x_3) \dots \mathbf{\Gamma P}(x_t) \\ &= \delta \mathbf{P}(x_1) \prod_{s=2}^t \mathbf{\Gamma P}(x_s) \\ &= (\alpha_t(1), \dots, \alpha_t(m)) \end{aligned}$$

for $t = 1, 2, \dots, T$. The name forward algorithm originates from the way of calculating α_t , i.e.

$$\begin{aligned} \alpha_0 &= \delta \mathbf{P}(x_1) \\ \alpha_t &= \alpha_{t-1} \mathbf{\Gamma P}(x_t) \text{ for } t = 1, 2, \dots, T. \end{aligned}$$

After a pass through all observations, the likelihood results from

$$L(\theta) = \alpha_T \mathbf{1}'.$$

In a similar way, the “backward algorithm” also permits the recursive computation of the likelihood, but starting with the last observation. To formulate the backward algorithm, let us define the vector β_t for $t = 1, 2, \dots, T$ so that

$$\begin{aligned}\beta'_t &= \mathbf{\Gamma P}(x_{t+1})\mathbf{\Gamma P}(x_{t+2}) \dots \mathbf{\Gamma P}(x_T) \dots \mathbf{1}' \\ &= \left(\prod_{s=t+1}^T \mathbf{\Gamma P}(x_s) \right) \mathbf{1}' \\ &= (\beta_t(1), \dots, \beta_t(m))\end{aligned}$$

The name backward algorithm results from the way of calculating β_t , i.e.

$$\begin{aligned}\beta_T &= \mathbf{1}' \\ \beta_t &= \mathbf{\Gamma P}(x_{t+1})\beta_{t+1} \text{ for } t = T-1, T-2, \dots, 1.\end{aligned}$$

Again, the likelihood can be calculated after a pass through all observations by

$$L(\theta) = \delta\beta_1.$$

In general, parameter estimation bases on the forward algorithm. The backward algorithm is, however, still useful because the quantities α_t and β_t together serve for a couple of interesting tasks. For example, they are the basis for deriving a particular type of conditional distributions and for state inference by local decoding (Zucchini et al., 2016, Ch. 5, pp. 81-93). We present details on local decoding at <https://github.com/timothée-bacri/hmm-tmb>. Last, it is well-known that the execution of the forward (or backward) algorithm may quickly lead to underflow errors, because many (for discrete distributions: all) elements of the vectors and matrices involved take values between zero and one. To avoid these difficulties, a scaling factor can be introduced. We follow the approach suggested by Zucchini et al. (2016, p. 48) and implement a scaled version of the forward algorithm, which directly provides the (negative) log-likelihood as result.

3.4 Reparameterization of the likelihood function

The representation of the likelihood and the algorithms shown above rely on the data and the set of parameters θ as input. The data are subject to several constraints:

- (i) Typically there are various constraints of the parameters in the conditional distribution. For the Poisson HMM, all elements of the parameter vector $\lambda = (\lambda_1, \dots, \lambda_m)$ must be non-negative.
- (ii) In general, the parameters γ_{ij} of the TPM $\mathbf{\Gamma}$ have to be non-negative, and the rows of $\mathbf{\Gamma}$ must sum up to one.

The constraints of the TPM can be difficult to deal with using constrained optimization of the likelihood. A common approach is to reparameterize the log-likelihood in terms of unconstrained “working” parameters $\{\mathbf{T}, \boldsymbol{\eta}\} = g^{-1}(\mathbf{\Gamma}, \boldsymbol{\lambda})$, as follows. A possible reparameterisation of $\mathbf{\Gamma}$ is given by

$$\gamma_{ij} = \frac{\exp(\tau_{ij})}{1 + \sum_{k \neq i} \tau_{ik}}, \text{ for } i \neq j$$

where τ_{ij} are $m(m-1)$ real-valued, thus unconstrained, elements of an m times m matrix \mathbf{T} with no diagonal elements. The diagonal elements of $\mathbf{\Gamma}$ follows implicitly from $\sum_j \gamma_{ij} = 1 \forall i$ (Zucchini et al., 2016, p. 51). The corresponding reverse transformation is given by

$$\tau_{ij} = \log \left(\frac{\gamma_{ij}}{1 - \sum_{k \neq i} \gamma_{ik}} \right) = \log(\gamma_{ij}/\gamma_{ii}), \text{ for } i \neq j$$

For the Poisson HMM the intensities can be reparameterised in terms of $\lambda_i = \exp(\eta_i)$, and consequently the unconstrained working parameters are given by $\eta_i = \log(\lambda_i)$, $i = 1, \dots, m$. Estimates of the "natural" parameters $\{\Gamma, \lambda\}$ can then be obtained by maximizing the reparameterised likelihood with respect to $\{\mathbf{T}, \eta\}$ and then transforming the estimated working parameters back to natural parameters via the above transformations, i.e. $\{\hat{\Gamma}, \hat{\lambda}\} = g(\hat{\mathbf{T}}, \hat{\eta})$. Note that in general the function g needs to be one-to-one for the above procedure to work.

4 Using TMB

In the following we show how parameter estimation for HMMs can be carried out efficiently via TMB. The TMB (R and C++) code used below is available from [https://github.com/\(TIMO:INSERT LINK TO REPOSITORY\)](https://github.com/(TIMO:INSERT LINK TO REPOSITORY)) to allow for consistent maintenance of the code.

4.1 Likelihood function

Similar to the linear regression example presented in 2.2, the first and essential step is to define our nll function to be minimized later in a suitable C++ file. In our case, this function calculates the negative log-likelihood presented by Zucchini et al. (2016, p. 48), and our C++ code is analog to the R-code shown by Zucchini et al. (2016, p. 331 - 333). This function, named *poi_hmm.cpp*, tackles our setting with conditional Poisson distributions only. An extension to for example Gaussian, binomial and exponential conditional distributions is straightforward. It only requires to modify the density function in the *poi_hmm.cpp* function and the related functions for parameter transformation presented in Section 3.4. We illustrate the implementation of these cases in the GitHub repository. However, note that the number of possible modelling setups is very large: e.g., the conditional distributions may vary from state to state, nested model specifications, the conditional mean may be linked to covariates, or the TPM could depend on covariates - to name only a few. Due to the very large number of possible extensions of the basic HMM, we refrain from implementing an R-package, but prefer to provide a proper guidance to the reader for building custom models suited to a particular application. As a small example, we illustrate how to implement a freely estimated initial distribution in the function *poi_hmm.cpp*. This modification can be achieved by uncommenting a couple of lines only.

4.2 Optimization

With the nll function available in C++, we can carry out the parameter estimation and all pre-/post-processing in R. In the following we describe the steps to be carried out.

- (i) Loading of the necessary packages, compilation of the nll function with TMB and subsequent loading, and loading of the auxiliary functions for parameter transformation.

```
# Load TMB and optimization packages
library(TMB)
library(optimr)
# Run the C++ file containing the TMB code
TMB::compile("code/poi_hmm.cpp")

## [1] 0

# Load it
dyn.load(dynlib("code/poi_hmm"))
# Load the parameter transformation function
source("functions/utils.R")
```


- (ii) Loading of the observations. The data are part of a large data set collected with the "TrackYourTinnitus" (TYT) mobile application, a detailed description of which is presented in Pryss et al. (2015a) and Pryss et al. (2015b) (TIMO: insert Pryss 2015a 2015b). We analyze 87 successive days of the "arousal" variable, which is measured on a discrete scale. Higher values correspond to a higher degree of excitement, lower values to a more calm emotional state (for details, see Probst et al., 2016).

TIMO: I need these references included (in the tex)

Loading the "arousal" variable can be achieved simply with

```
load("data/tinnitus.RData")
```

TIMO: update the table as well please

Table 1 presents the raw data, which are also available for download at the GitHub repository.

6	5	3	6	4	3	5	6	6	6	4	6	6	4	6	6	6	6	4	6	5	6	7	6	5	5	7	6	5	6	5	6	6	5	6	7	7	6	7	6	6	6	5	7	6	1	6	0	2	1	6	7	6	6	6	5	5	6	6	2	5	0	1	1	1	2	3	1	3	1	3	0	1	1	4	1	4	1	2	2	2	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Table 1 Tinnitus data

- (iii) Initialization of the number of states and starting (or initial) values for the optimization. First, the number of states needs to be determined. As explained by Pohle et al. (2017a), Pohle et al. (2017b), and Zucchini et al. (2016, Section 6) (to name only a few), usually one would first fit models with a different number of states. Then, these models are evaluated e.g. by means of model selection criteria (as carried out by Leroux and Puterman, 1992) or prediction performance (Celeux and Durand, 2008). Since the results reported by Leroux and Puterman (1992) show that a two-state model is preferred by the BIC, we focus on this model only here - although other choices would be possible, e.g. the AIC selects a three-state model. The list object `TMB_data` contains the data and the number of states.

```
# Model with 2 states
m <- 2
TMB_data <- list(x = tinn_data, m = m)
```

Secondly, initial values for the optimization procedure need to be defined. Although we will apply unconstrained optimization, we initialize the natural parameters, because this is much more intuitive and practical than handling the working parameters.

```
# Generate initial set of parameters for optimization
lambda <- c(1, 3)
gamma <- matrix(c(0.8, 0.2,
                  0.2, 0.8), byrow = TRUE, nrow = m)
```

- (iv) Transformation from natural to working parameters. The previously created initial values are transformed and stored in the list `parameters` for the optimization procedure.

```
# Turn them into working parameters
parameters <- pois.HMM.pn2pw(m, lambda, gamma)
```

- (v) Creation of the TMB negative log-likelihood function with its derivatives. This object, stored as `obj_tmb` requires the data, the initial values, and the previously created the DLL as input. Setting argument `silent = TRUE` disables tracing information and is only used here to avoid excessive output.

```
obj_tmb <- MakeADFun(TMB_data, parameters,
                     DLL = "poi_hmm", silent = TRUE)
```

This object also contains the previously defined initial values as a vector (`par`) rather than a list. The negative log-likelihood (`fn`), its gradient (`gr`), and Hessian (`he`) are functions of the parameters (in vector form) while the data are considered fixed:

```
obj_tmb$par

##      tlambda      tlambda      tgamma      tgamma
##  0.000000  1.098612 -1.386294 -1.386294

obj_tmb$fn(obj_tmb$par)

## [1] 228.3552

obj_tmb$gr(obj_tmb$par)

##           [,1]      [,2]      [,3]      [,4]
## [1,] -3.60306 -146.0336  10.52832 -1.031706

obj_tmb$he(obj_tmb$par)

##           [,1]      [,2]      [,3]      [,4]
## [1,]  1.902009 -5.877900 -1.3799682  2.4054017
## [2,] -5.877900 188.088247 -4.8501589  2.3434284
## [3,] -1.379968 -4.850159  9.6066700 -0.8410438
## [4,]  2.405402  2.343428 -0.8410438  0.7984216
```

- (vi) Execution of the optimization. For this step we rely again on the optimizer implemented in the `nlminb` function. The arguments, i.e. initial values for the parameters and the function to be optimized, are extracted from the previously created TMB object.

```
mod_tmb <- nlminb(start = obj_tmb$par, objective = obj_tmb$fn)
# Check that it converged successfully
mod_tmb$convergence == 0

## [1] TRUE
```

There are alternatives to `nlminb`, but we focus on it here because of its good performance in terms of speed. Further details are available in Section ?? Table 6.4.

- (vii) Obtaining the ML estimates of the natural parameters together with their standard errors is possible by using the previously introduced command `sdreport`. Recall that this requires the parameters of interest to be treated by the `ADREPORT` statement in the C++ part. It should be noted that the presentation of the set of parameters `gamma` below results from a column-wise representation of the TPM.

```
summary(sdreport(obj_tmb, par.fixed = mod_tmb$par), "report")

##           Estimate Std. Error
## lambda 1.63641070 0.27758294
## lambda 5.53309626 0.31876141
## gamma  0.94980192 0.04374682
## gamma  0.02592209 0.02088689
## gamma  0.05019808 0.04374682
## gamma  0.97407791 0.02088689
## delta  0.34054163 0.23056401
## delta  0.65945837 0.23056401
```

Note that the table above also contains estimation results for δ and accompanying standard errors, although δ is not estimated, but derived from Γ . We provide further details on this aspect in Section 5.1. The value of the nll function in the minimum found by the optimizer can also be extracted directly from the object `mod_tmb` by accessing the list element `objective`:

```
mod_tmb$objective

## [1] 168.5361
```

- (viii) In the optimization above we already benefited from an increased speed due to the evaluation of the nll in C++ compared to the forward algorithm being executed entirely in R. However, the use of TMB also permits to introduce the gradient and/or the Hessian computed by TMB into the optimization procedure. This is in general advisable, because TMB provides an exact value of both gradient and Hessian up to machine precision, which is superior to approximations used by optimizing procedure. Similar to the nll, both quantities can be extracted directly from the TMB object `obj_tmb`:

```
# The negative log-likelihood is accessed by the objective
# attribute of the optimized object
mod_tmb <- nlminb(start = obj_tmb$par, objective = obj_tmb$fn,
                  gradient = obj_tmb$gr, hessian = obj_tmb$he)
mod_tmb$objective

## [1] 168.5361
```

Note that passing the exact gradient and Hessian as provided by TMB to `nlminb` leads to the same minimum, i.e. value of the nll function, here.

On a minor note, when comparing our estimation results to those reported by Leroux and Puterman (1992), some non-negligible differences can be noted. The reasons for this are difficult to determine, but some likely explanations are given in the following. First, differences in the parameter estimates may result e.g. from the optimizing algorithms used and related setting (e.g. convergence criterion, number of

steps, optimization routines used in 1992,...). Moreover, Leroux and Puterman (1992) seem to base their calculations on an altered likelihood, which is reduced by removing the constant term $\sum_{i=1}^T \log(x_i!)$ from the log-likelihood. This modification may also possess an impact on the behavior of the optimization algorithm, as e.g. relative convergence criteria and step size could be affected.

4.3 Basic nested model specification

In the context of HMMs (and other statistical models), nested models or models subject to certain parameter restrictions are commonly used. For example, it may be necessary to fix some parameters because of biological or physical constraints. TMB can be instructed to treat selected parameters as constants, or impose equality constraints on a set of parameters. For the practical implementation, it is noteworthy that such parameter restrictions should be imposed on the working parameters. However, it is also easily possible to impose restrictions on a natural parameter (e.g. λ), and then identify the corresponding restriction on the working parameter (i.e. $\log(\lambda)$).

TIMO: lets chat about the following. Fixing a value in the transition probability matrix might be possible, but is clearly troublesome given the lack of a one-to-one correspondence with the natural parameters (one parameter of the natural TPM doesn't correspond to one parameter of the working TPM but multiple), so we fix a Poisson mean instead.

We illustrate a simple nested model specification by fixing λ_1 to one in our two-state Poisson HMM, the other parameter components correspond to the previous initial values.

```
# Get the previous values, and fix some
fixed_par_lambda <- lambda
fixed_par_lambda[1] <- 1
```

We then transform these natural parameters into a set of working parameters.

```
# Transform them into working parameters
new_parameters <- pois.HMM.pn2pw(m = m,
                                lambda = fixed_par_lambda,
                                gamma = gamma)
```

For instructing TMB to treat selected parameters as constants, the `map` argument of the `MakeADFun` has to be specified in addition to the usual arguments. The `map` argument is a list consisting factor-valued vectors which possess the same length as the working parameters and carry their names as well. The factor levels have to be unique for the regular parameters not subject to specific restrictions. If a parameter is fixed the corresponding entry of the `map` argument is filled with `NA`. To impose equality constraints (e.g. $\lambda_1 = \lambda_2$), the corresponding factor level has to be identical for the concerned entries. In our example, this leads to:

```
map <- list(tlambda = as.factor(c(NA, 1)),
            tgamma = as.factor(c(2, 3)))
fixed_par_obj_tmb <- MakeADFun(TMB_data, new_parameters,
                              DLL = "poi_hmm",
                              silent = TRUE,
                              map = map)
```

Estimation of the remaining model parameters and extraction of the results is achieved as before.

```

fixed_par_mod_tmb <- nlminb(start = fixed_par_obj_tmb$par,
                           objective = fixed_par_obj_tmb$fn,
                           gradient = fixed_par_obj_tmb$gr,
                           hessian = fixed_par_obj_tmb$he)
summary(sdreport(fixed_par_obj_tmb), "report")

##           Estimate Std. Error
## lambda 1.00000000 0.00000000
## lambda 5.50164872 0.30963641
## gamma 0.94561055 0.04791050
## gamma 0.02655944 0.02133283
## gamma 0.05438945 0.04791050
## gamma 0.97344056 0.02133283
## delta 0.32810136 0.22314460
## delta 0.67189864 0.22314460

```

Note that the standard error of λ_1 equals zero, because it is no longer considered a parameter and does not enter the optimization procedure.

4.4 State inference and forecasting

After estimating a HMM by the procedures illustrated in Section 4.2, it is possible to carry out a couple analyses that provide insight into the interpretation of the estimated model. These include, e.g., the so-called smoothing probabilities, which correspond to the probability of being in state i at time t for $i = 1, \dots, m$, $t = 1, \dots, n$, given all observations. These probabilities can be obtained by

$$P(C_t = i | X^{(n)} = x^{(n)}) = \frac{\alpha_t(i)\beta_t(i)}{L(\hat{\theta})},$$

where $\hat{\theta}$ denotes the set of ML estimates. The derived smoothing probabilities then serve for determining the most probable state i_t^* at time t given the observations by

$$i_t^* = \arg \max_{i_t \in \{1, \dots, m\}} P(C_t = i_t | X^{(n)} = x^{(n)}).$$

Furthermore, the Viterbi algorithm determines the overall most probable sequence of states i_1^*, \dots, i_T^* , given the observations. This is achieved by evaluating

$$(i_1^*, \dots, i_n^*) = \arg \max_{i_1, \dots, i_n \in \{1, \dots, m\}} P(C_1 = i_1, \dots, C_n = i_n | X^{(n)} = x^{(n)}).$$

Other quantities of interest include the forecast distribution or h -step-ahead probabilities, which are obtained through

$$P(X_{n+h} = x | X^{(n)} = x^{(n)}) = \frac{\alpha_n \Gamma^h \mathbf{P}(x) \mathbf{1}'}{\alpha_n \mathbf{1}'} = \varphi_n \Gamma^h \mathbf{P}(x) \mathbf{1}',$$

where $\varphi_n = \alpha_n / \alpha_n \mathbf{1}'$.

All the quantities shown above and the related algorithms for deriving them are described in detail in Zucchini et al. (2016, Chapter 5). In order to apply these algorithms, it is only necessary to extract the quantities required as input from a suitable `MakeADFun` object. Note that most algorithms rely on scaled versions of the forward- and backward-algorithm. This is illustrated in detail on GitHub.

5 Confidence intervals

A common approach for deriving confidence intervals (CIs) for the estimated parameters of statistical models bases on finite-difference approximations of the Hessian. This technique is, however, not suited for most HMMs due to computational difficulties, as already pointed out by Visser *et al.* (2000). The same authors suggest likelihood profile CIs or bootstrap based CIs as potentially better alternatives. Despite the potentially high computational load, bootstrap based CIs have become an established method in the context of HMMs (Bulla and Berzel, 2008; Zucchini *et al.*, 2016) and found widespread application by practitioners.

In this section we illustrate how CIs based on the Hessian, likelihood profiling, and the bootstrap can be efficiently implemented by integrating TMB. This permits in particular to obtain Hessian based and likelihood profile based CIs at very low computational cost. For simplicity, we illustrate our procedures by means of the parameter λ_2 of our two-state Poisson HMM. We will further address the resulting CIs for Γ and λ and performance-related aspects in Section 6.

5.1 Hessian based confidence intervals

Since the negative log-likelihood function of HMMs typically depends on the working parameters, evaluation of the Hessian in the optimum found by numerical optimization only serves for inference about the working parameters. From a practical perspective, however, inference about the natural parameters usually is of interest. As the Hessian $\nabla^2 \log L(\{\hat{\mathbf{T}}, \hat{\boldsymbol{\eta}}\})$ refers to the working parameters $\{\mathbf{T}, \boldsymbol{\eta}\}$, the delta method is suitable to obtain an estimate of the covariance matrix of $\{\hat{\mathbf{T}}, \hat{\boldsymbol{\lambda}}\}$ by

$$\Sigma_{\hat{\mathbf{T}}, \hat{\boldsymbol{\lambda}}} = -\nabla g(\hat{\mathbf{T}}, \hat{\boldsymbol{\eta}}) \left(\nabla^2 \log L(\hat{\mathbf{T}}, \hat{\boldsymbol{\eta}}) \right)^{-1} \nabla g(\hat{\mathbf{T}}, \hat{\boldsymbol{\eta}})', \quad (2)$$

with $\{\hat{\mathbf{T}}, \hat{\boldsymbol{\lambda}}\} = g(\hat{\mathbf{T}}, \hat{\boldsymbol{\eta}})$ as defined in Section 3.4. From a user's perspective, it is highly convenient that the entire right-hand side of Equation 2 can be directly computed via automatic differentiation in TMB. Moreover, it is particularly noteworthy that the standard errors of derived parameters can be calculated by the delta-method similarly. For example, the stationary distribution $\boldsymbol{\delta}$ is a function of Γ in our case, and TMB provides a straightforward way to obtain standard errors of $\boldsymbol{\delta}$. This is achieved by first defining $\boldsymbol{\delta}$ inside the C++ file `poi_hmm.cpp` (or, in our implementation, the related `utils.cpp`, which gathers auxiliary functions). Secondly, it is necessary to call `ADREPORT` on $\boldsymbol{\delta}$ within the `poi_hmm.cpp` file. To display the resulting estimates and corresponding standard errors in R, one can rely on the command shown previously in Section 4.2.

Subsequently, Wald-type confidence intervals (Wald, 1943) follow in the usual manner. For example, the $(1 - \alpha)\%$ CI for λ_1 is given by $\lambda_1 \pm z_{1-\alpha/2} * \sigma_{\lambda_1}$ where z_x is the x -percentile of the standard normal distribution, and σ_{λ_1} is the standard error of λ_1 obtained via the delta method. This part is easily implemented in R. We illustrate the calculation of these CIs for our two-state Poisson HMM on GitHub.

Finally, note that the reliability of Wald-type CIs may suffer from a singular Fisher information matrix, which can occur for many different types of statistical models, including HMMs. This also jeopardizes the validity of AIC and BIC criteria. For further details on this topic, see e.g. Drton and Plummer (2016).
TIMO: Insert the reference below here.

TIMO: All this on Github only

The $100(1 - \alpha)\%$ confidence interval for

```

adrep <- summary(sdreport(obj_tmb), "report")

# Get the 97.5 percentile of the standard normal distribution
q95_norm <- qnorm(1 - 0.05 / 2)

# Create the confidence interval
# Extract the values
estimates <- adrep[, "Estimate"]
std_errors <- adrep[, "Std. Error"]
# Create the bounds
lower_bound <- estimates - q95_norm * std_errors
upper_bound <- estimates + q95_norm * std_errors
# Show the CI
cbind(lower_bound, upper_bound)

##           lower_bound upper_bound
## lambda  1.09235840   2.18046360
## lambda  4.90833475   6.15785677
## gamma   0.86406003   1.03554416
## gamma  -0.01501550   0.06685957
## gamma  -0.03554416   0.13593997
## gamma   0.93314043   1.01501550
## delta  -0.11135586   0.79243986
## delta   0.20756014   1.11135586

```

5.2 Likelihood profile based confidence intervals

The Hessian based CIs presented above rely on asymptotic normality of the ML estimator. Properties of the ML estimator may, however, change in small samples. Moreover, symmetric CIs may not be suitable if the ML estimator lies close to a boundary of the parameter space. This occurs, e.g., when states are highly persistent, which leads to entries close to one in the TPM. An alternative approach to construct CIs bases on the so-called profile likelihood (see, e.g., Venzon and Moolgavkar, 1988; Meeker and Escobar, 1995), which has also shown a satisfactory performance in the context of HMMs (Visser et al., 2000).

In the following, we illustrate the principle of likelihood profile based CIs by the example of the parameter λ_2 in our two-state Poisson HMM. The underlying basic idea is to identify those values of our parameter of interest λ_2 in the neighborhood of $\hat{\lambda}_2$ that lead to a significant change in the log-likelihood, whereby the other parameters (i.e. Γ, λ_1) are considered nuisance parameters (Meeker and Escobar, 1995). The "term nuisance parameters" means that these parameters need to be re-estimated (by maximizing the likelihood) for any fixed value of λ_2 different to $\hat{\lambda}_2$. That is, the profile likelihood of λ_2 is defined as

$$L_p(\lambda_2) = \max_{\Gamma, \lambda_1} L(\Gamma, \lambda)$$

In order to construct profile likelihood based CIs, let $\{\hat{\Gamma}, \hat{\lambda}\}$ denote the ML estimate for our HMM computed as described in Section 4.2. Evaluation of the log-likelihood function in this point results in the value $\log L(\{\hat{\Gamma}, \hat{\lambda}\})$. The deviation of the likelihood of the ML estimate and the profile likelihood in the point λ_2^p is then captured by the following likelihood ratio:

$$R_p(\lambda_2) = -2 \left[\log(L_p(\lambda_2)) - \log(L(\hat{\Gamma}, \hat{\lambda})) \right] \quad (3)$$

As described above, the log-likelihood $\log(L_p(\lambda_2))$ results from re-estimating the two-state HMM with fixed parameter λ_2 . Therefore, this model effectively corresponds to a nested model of the full model with ML estimate $\hat{\Gamma}, \hat{\lambda}$. Consequently, R_p asymptotically follows a χ^2 distribution with one degree of freedom - the difference in degrees of freedom between the two models. Based on this, a CI for λ_2 can be derived by evaluating R_p at many different values of λ_2^p and determining when the resulting value of R_p becomes "too extreme". That is, for a given α , one needs to calculate the $1 - \alpha$ quantile of the χ^2_1 distribution (e.g., 3.841 for $\alpha = 5\%$). The CI at level $1 - \alpha$ for the parameter λ_2 is then given by

$$\left\{ \lambda_2 : R_p(\lambda_2) < \chi^2_{1,(1-\alpha)} \right\} \quad (4)$$

For simplicity, the principles of likelihood profiling shown above rely on the natural parameters. Our nll function is, however, parameterized in terms of and optimized with respect to the working parameters. In practice, this aspect is easy to deal with. Once a profile CI for the working parameter (here η_2) has been obtained following the procedure above, the corresponding CI for the natural parameter λ_2 results directly from transforming the upper and lower boundary of the CI for η_2 by the one-to-one transformation $\lambda_2 = \exp(\eta_2)$. Basis for this transformation is the invariance property of ML estimation, as described e.g. by Casella and Berger (2021, Theorem 7.2.10 on p. 320)TIMO: insert a reference to Casalla Berger Section?.

TMB provides an easy way to profiling through the function `tmbprofile`, which requires several inputs. First, the well-known `MakeADFun` object called `obj_tmb` from our two-state Poisson HMM. Secondly, the position of the (working) parameter to be profiled via the `name` argument. This position refers to the position in the parameter vector `obj_tmb$par`. Moreover, here the optional `trace` argument indicates how much information on the optimization is displayed. The following commands permit to profile the second working parameter $\eta_2 = \log(\lambda_2)$.

```
profile <- tmbprofile(obj = obj_tmb,
                     name = 2,
                     trace = FALSE)
plot(profile, level = 0.95)
```

TIMO: we need this figure in an includefigure environment such that we can label / refer to it
JAN: look at tex file here

Furthermore, Figure 1 obtained via the `plot` function shows the resulting profile nll as a function of the working parameter η_2 . The vertical and horizontal lines correspond to the boundaries of the confidence interval and the critical value of the nll derived from Equation 4, respectively. The CI for η_2 can directly be extracted via the function `confint`:

```
# Confidence interval of tlambda
confint(profile, level = 0.95)

##               lower      upper
## tlambda 1.593141 1.820641
```

The corresponding CI for λ_2 the follows from:

```
# Confidence interval of lambda
exp(confint(profile, level = 0.95))

##               lower      upper
## tlambda 4.919178 6.175815
```

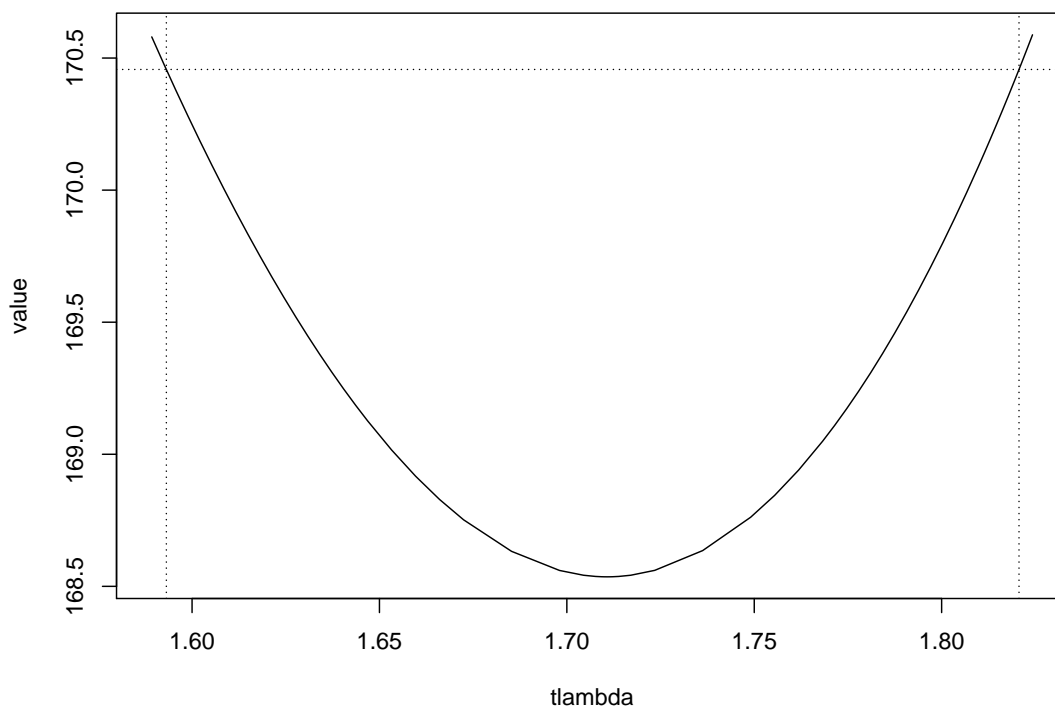



Figure 1 Profile likelihood

While simple linear combinations of variables can be profiled through the argument `lincomb` in the `tmbprofile` function, this is not possible for more complex functions of the parameters. This includes the stationary distribution δ , for which CIs cannot be obtained by this method.

Last, note that the function `tmbprofile` carries out several optimization procedures internally for calculating profile CIs. If this approach fails, or prefers a specific optimization routine, the necessary steps for profiling can also be implemented by the user. To do so, it would be - roughly speaking - necessary to compute $R_p(\eta_2)$ for a sufficient number of η_2 values to achieve the desired precision.

TIMO: script on Github showing how to do it for the elements of the TPM as 2nd section / additional code - check this part with Geir

5.3 Bootstrap based confidence intervals

The last approach for deriving CIs is the bootstrap, which is frequently applied by many practitioner. Efron and Tibshirani (1993) describe the underlying concepts of the bootstrap in their seminal manuscript. Many different bootstrap techniques have evolved since then, leading to an extensive treatment of this subject in the scientific literature.

A thorough overview of this subject would go beyond the scope of this paper. As pointed out by Härdle et al. (2003), the so-called parametric bootstrap is suitable in the context of time series models. For further details on the bootstrap for HMMs including the implementation of a parametric bootstrap, we refer to

Zucchini et al. (2016, Ch. 3, pp. 56-60).

Basically all versions of the bootstrap have in common that some kind of re-sampling procedure needs to be carried out first. Secondly, the model of interest is re-estimated for each of the re-sampled data sets. A natural way to accelerate the second part consists in the use of `tmb` for the model estimation by means of the procedures presented in Section 4.2. Our GitHub page contains a detailed example illustrating the implementation of a parametric percentile bootstrap for our 2-state Poisson HMM.

All the following to Github

JAN: What came here is now commented out, and on GitHub

TIMO: parametric BS, not non-parametric BS. right?!

6 Application to different data sets

The aim of this section is to demonstrate the performance of `tmb` by means of a couple of practical examples that differ in terms of the number of observations and model complexity. These examples include the TYT data shown above, a data set of fetal lamb movements and one of hospital visits, and simulated data sets. For the performance comparisons, focus lies on computational speed and the reliability of confidence intervals. The R-scripts necessary for this section may serve interested users for investigating their own HMM-setting, and are all available on GitHub.

6.1 TYT data

We begin by investigating the speed of five approaches for parameter estimation: one without the usage of TMB, and four with TMB. In the following, DM denotes direct maximization of the log-likelihood through the optimization function `nlminb` without TMB. Furthermore, TMB , TMB_H , TMB_G , and TMB_{GH} denote direct maximization with TMB without making use of the exact gradient and Hessian that TMB can provide, with the Hessian, with the gradient, and with both gradient and Hessian, respectively.

As preliminary reliability check of our IT infrastructure and setup, we timed the fitting of our 2-state HMM to the TYT data with the help of the `microbenchmark` package. For this data set, all five approaches converged to the same maximum apart from minor deviations typical for numerical optimization (see Table 2).

TIMO: can you add the `nllk` at the bottom of the table? And is it possible to increase the number of digits such that a small difference between the estimates becomes visible? I would be interested to see how to change the digits myself as well...in your table function I cannot find a rounding option

	DM	TMB	TMB_G	TMB_H	TMB_{GH}
λ_1	1.64	1.64	1.64	1.64	1.64
λ_2	5.53	5.53	5.53	5.53	5.53
$\gamma_{1,1}$	0.95	0.95	0.95	0.95	0.95
$\gamma_{2,1}$	0.03	0.03	0.03	0.03	0.03
$\gamma_{1,2}$	0.05	0.05	0.05	0.05	0.05
$\gamma_{2,2}$	0.97	0.97	0.97	0.97	0.97
δ_1	0.34	0.34	0.34	0.34	0.34
δ_2	0.66	0.66	0.66	0.66	0.66

Table 2 Estimates of 2 state Poisson HMM with and without using TMB, estimated on the tinnitus dataset.

Table 3 shows the resulting average time required for the parameter estimation and the number of iterations needed by each approach, measured over 100 replications. The results show that the use of TMB significantly accelerates parameter estimation in comparison with *DM*. The strongest acceleration is achieved by *TMB_G*, underlining the benefit of using the gradient provided by TMB. Moreover, *TMB_{GH}* requires less iterations than the other approaches. However, the approximation of the Hessian seems to increase the computational burden.


	<i>DM</i>	<i>TMB</i>	<i>TMB_G</i>	<i>TMB_H</i>	<i>TMB_{GH}</i>
Time (ms)	23.9 (23.5, 24.4)	1.9 (1.77, 2.02)	0.868 (0.857, 0.88)	1.78 (1.77, 1.8)	2.39 (2.19, 2.59)
Iterations	13	13	13	13	7

Table 3 Average duration (in milliseconds) together with 95% CI and number of iterations required for fitting a 2-state Poisson HMM to the TYT data. The CIs are of Wald-type and base on the standard error of the mean derived from 100 replications.

Next, we verified the reproducibility of the acceleration by TMB in a parametric bootstrap setting. More specifically, we simulated 100 bpptstrap samples from the model estimated on the TYT data. Then, we re-estimated the same model by our five approaches and derived acceleration ratios (with *DM* as reference approach) and their corresponding percentile CIs. As shown in Table 4, all acceleration ratios take values significantly larger than one, whether the gradient and Hessian are passed from TMB or not. In addition, the findings from the single TYT data set are confirmed with *TMB_G* providing the strongest acceleration.

	<i>TMB</i>	<i>TMB_G</i>	<i>TMB_H</i>	<i>TMB_{GH}</i>
Acceleration ratio	14.5 (12.8, 15.7)	28.7 (25.3, 31.6)	14.4 (12.8, 15.7)	11.4 (9.21, 14.6)
Iterations	15.6 (13.5, 19)	15.6 (13.5, 19)	15.6 (13.5, 19)	6.85 (6, 9)

Table 4 Acceleration ratios together with 95% CIs when using TMB in a bootstrap setting with 100 bootstrap samples

For some bootstrap samples, one or several of the estimation algorithms did not converge properly. In such cases, we discarded the results, generated an additional bootstrap sample, and re-ran the parameter estimation. Convergence problems mostly occurred due to *TMB* and *TMB_H* failing. We therefore recommend to pass at least the gradient when optimizing with TMB for increased stability of the parameter estimation with *nlminb*. As additional check of the acceleration obtained by TMB, we also timed the computation of log-likelihood alone in the bootstrap setting. The acceleration factor of TMB compared to conventional R code was estimated as 10.6 (6.  4.3). This underlines that the acceleration is obtained by two factors: the use of C++ code on the one hand, and computation of the gradient and/or Hessian by TMB on the other hand.

TIMO: In Table 5 it would be good to get percent values for the coverage probabilities, with one digit after the comma. Or three digits after the comma. Geir might have an opinion here as well.

JAN: For profile CI, I used `TMB::tmbprofile` and `confint`. For bootstrap, percentile CI. For TMB CI, estimate $\pm 1.96 * \text{std_error}$.

The negative values are expected for estimates close to 0, because the interval is Wald-type. Similarly, $\gamma_{2,2}$ has an upper bound of 1.04, above 1. Is it normal to manually take out impossible values? We can also switch to percentile CIs to prevent this.

I used the same technique for all simulations: generate a dataset as long as TMB or `marqLevAlg` fail to

converge,

TIMO: it is normal to remove impossible values / cap them off at their natural bound. It is good to use Wald-type intervals, because those are the ones which we described in the section on CIs - so it would be strange to use something different here.

Last, we investigate CIs obtained for the TYT data by the three different methods described in Table 5, TMB_{GH} served as sole estimation approach. The columns to the left of ?? show the parameter estimates and 95% CIs obtained via the methods described in Section 5. That is, the Hessian provided by TMB, likelihood profiling, and bootstrapping. For this data set, no major differences between the different CIs are visible. Furthermore, in we assessed the accuracy of the different CIs by coverage probabilities. For calculating these coverage probabilities, we used a Monte Carlo setting similar the one described above with 100 replications. Samples for which the estimation algorithm did not converge were replaced. The results, shown on the right of Table 5 indicate that all methods provide comparably reliable CI estimates. Only the coverage probabilities for the parameters δ_1 and δ_2 take values deviating considerably from the 95% level.


Parameter	Estimate	 CI		Profile CI		Bootstrap CI		Coverage prob.		
		L.	U.	L.	U.	L.	U.	TMB	Profile	Bootstrap
λ_1	1.64	1.09	2.18	1.15	2.23	1.29	2.12	0.96	0.96	0.99
λ_2	5.53	4.91	6.16	4.92	6.18	5.18	5.97	0.97	0.97	0.97
$\gamma_{1,1}$	0.95	0.86	1.04	0.82	1.00	0.78	0.98	0.96	0.94	0.94
$\gamma_{2,1}$	0.03	-0.02	0.07	0.00	0.09	0.01	0.07	0.93	0.95	0.95
$\gamma_{1,2}$	0.05	-0.04	0.14	0.00	0.18	0.02	0.22	0.96	0.94	0.94
$\gamma_{2,2}$	0.97	0.93	1.02	0.91	1.00	0.93	0.99	0.93	0.95	0.95
δ_1	0.34	-0.11	0.79			0.08	0.64	0.86		0.95
δ_2	0.66	0.21	1.11			0.36	0.92	0.86		0.95

Table 5 CIs for the TYT dataset. From left to right, the columns contain: the parameter name, parameter estimate, and lower (L.) and upper (U.) bound of the corresponding 95% CI derived by TMB, likelihood profiling, and percentile bootstrap. Then follow coverage probabilities derived for these three methods in a Monte-Carlo study.

BOOTSTRAP_SAMPLES=500 replications for bootstrap CIs

COVERAGE_SAMPLES=200 replications for coverage CIs

BENCHMARK_SAMPLES=100 replications for the benchmarking.

6.2 Lamb data

We chose the well-known data set presented in Leroux and Puterman (1992). It consists of the number of movements by a fetal lamb in 240 consecutive 5-second intervals.

The counts are available in the supporting information.

(2) hidden states were specified in the HMM estimated on the lamb dataset,

We timed estimations of HMMs using different parameters on a dataset provided by Leroux and Puterman (1992).

Speed of TMB

TMB accelerates the estimation time for the HMMs, as with the hospital dataset.

The average time ratios from using TMB on HMMs estimated on the lamb dataset are all above 1 (Table 6), thus showing that estimation using TMB is faster than a regular estimation procedure without using TMB even on a small dataset.

m	<i>TMB</i>	<i>TMB_G</i>	<i>TMB_H</i>	<i>TMB_{GH}</i>
2	21.6 (19, 23.6)	42.7 (33.2, 54.5)	21.8 (19.4, 23.8)	13.2 (9.62, 24.9)

Table 6 Acceleration when using TMB for m state Poisson HMM parameter estimation, estimated on the lamb dataset. Each procedure was repeated and timed once on 100 sample datasets. With 2 hidden states, *TMB_G* accelerated the estimation by an average factor of 42, with a 95% quantile confidence interval of 7 (33.2,

m	<i>DM</i>	<i>TMB</i>	<i>TMB_G</i>	<i>TMB_H</i>	<i>TMB_{GH}</i>
2	3.1 (2.85, 3.87)	1.52 (1.38, 1.99)	3.07 (2.84, 3.93)	4.23 (3.88, 5.57)	70.1 (62.8, 84.3)

Table 7 Duration (in millisecond) of m state Poisson HMM parameter estimation on the lamb dataset. Each procedure was timed 100 times. This table summarizes the computer's performance across multiple attempts at the same computation, in order to verify how reliable timing an estimation is. With 2 hidden states, estimation with *TMB_G* took an average 3.0 ms to compute, with an empirical 95% quantile confidence interval of 7 (2.84, 3.

Optimizing with TMB gives the same estimates as optimizing without (Table 8) when estimating a 2 state Poisson HMM on the lamb dataset.

	<i>DM</i>	<i>TMB</i>	<i>TMB_G</i>	<i>TMB_H</i>	<i>TMB_{GH}</i>
λ_1	0.26	0.26	0.26	0.26	0.26
λ_2	3.11	3.11	3.11	3.11	3.11
$\gamma_{1,1}$	0.99	0.99	0.99	0.99	0.99
$\gamma_{2,1}$	0.31	0.31	0.31	0.31	0.31
$\gamma_{1,2}$	0.01	0.01	0.01	0.01	0.01
$\gamma_{2,2}$	0.69	0.69	0.69	0.69	0.69
δ_1	0.96	0.96	0.96	0.96	0.96
δ_2	0.04	0.04	0.04	0.04	0.04

Table 8 Estimates of 2 state Poisson HMM with and without using TMB, estimated on the lamb dataset.

Moreover, the acceleration ratios from using TMB (Table 9 and Table 12) are all above 1, thus showing that on a comparably small dataset, the objective function computation can be accelerated.

m	<i>TMB</i>	<i>TMB_G</i>	<i>TMB_H</i>	<i>TMB_{GH}</i>
2	19.1 (13.1, 24.2)	18.8 (13.4, 24.5)	18.2 (10.7, 24.5)	19.1 (13.3, 23.7)

Table 9 Acceleration when using TMB for m state Poisson HMM negative log-likelihood calculation, estimated on the lamb dataset. Each procedure was repeated and timed once on 100 sample datasets. With 2 hidden states, *TMB_G* accelerated the estimation by an average factor of 18, with a 95% quantile confidence interval of 8 (13.4,

6.3 Simulation study

(2, 3) for the simulated dataset,

The second dataset is a sequence of random numbers generated by a m -state Poisson HMM. We used the code below to generate that simulated dataset, with $m = 2, 3$.

```
DATA_SIZE_SIMU <- 2000
m <- 2
# Generate parameters
lambda <- seq(1, 7, length.out = m)
# Create the transition probability matrix with 0.8 on its diagonal
gamma <- matrix(0.2 / (m - 1), nrow = m, ncol = m)
diag(gamma) <- 0.8
delta <- stat.dist(gamma)

#simulate the data
simul_data <- pois.HMM.generate_sample(ns = DATA_SIZE_SIMU,
                                       mod = list(m = m,
                                                  lambda = lambda,
                                                  gamma = gamma,
                                                  delta = delta))
```

The average speed increases using a simulated dataset (??) shows similar results to the speed gains from the other datasets, showing that TMB is useful for estimating Poisson HMMs on medium sized datasets.

m	TMB	TMB_G	TMB_H	TMB_{GH}
2	15.5 (14.2, 16.8)	30.7 (24.7, 37.7)	15.5 (13.6, 16.5)	18.2 (12.8, 22.7)
3	11.6 (10.6, 12.4)	45.4 (36.7, 55.6)	11.6 (10.6, 12.4)	22.1 (14.2, 28.5)

Table 10 Acceleration when using TMB for m state Poisson HMM parameter estimation, estimated on the simulated dataset. Each procedure was repeated and timed once on 100 sample datasets. With 2 hidden states, TMB_G accelerated the estimation by an average factor of 30, with a 95% quantile confidence interval of 7 (24.7,

m	DM	TMB	TMB_G	TMB_H	TMB_{GH}
2	46 (44, 48)	19 (18, 20)	46 (44, 48)	14 (14, 15)	853 (831, 1001)
3	206 (202, 211)	59 (57, 81)	207 (202, 217)	57 (56, 59)	2584 (2521, 2737)

Table 11 Duration (in millisecond) of m state Poisson HMM parameter estimation on the simulated dataset. Each procedure was timed 100 times. This table summarizes the computer's performance across multiple attempts at the same computation, in order to verify how reliable timing an estimation is. With 2 hidden states, estimation with TMB_G took an average 46 ms to compute, with an empirical 95% quantile confidence interval of (44, 48)

The acceleration ratios displayed in Table 10 are all above 1, thus showing that on medium sized datasets, the estimation computation can be accelerated.

Moreover, the acceleration ratios from using TMB (Table 12) are all above 1, thus showing that on a medium sized datasets, the objective function computation can be accelerated.

Similar conclusions are obtained for Poisson HMMs estimated on the hospital dataset and simulated datasets.

6.4 Hospital data

and (2, 3) for the hospital dataset) to compare speeds.

m	<i>TMB</i>	<i>TMB_G</i>	<i>TMB_H</i>	<i>TMB_{GH}</i>
2	16 (14, 19)	16 (13, 19)	16 (13, 19)	16 (12, 18)
3	11 (9, 12)	11 (9, 13)	11 (9, 13)	11 (9, 13)

Table 12 Acceleration when using TMB for m state Poisson HMM negative log-likelihood calculation, estimated on the simulated dataset. Each procedure was repeated and timed once on 100 sample datasets. With 2 hidden states, *TMB_G* accelerated the estimation by an average factor of 16, with a 95% quantile confidence interval of (13, 19)

m	<i>TMB</i>	<i>TMB_G</i>	<i>TMB_H</i>	<i>TMB_{GH}</i>
2	14 (12, 15)	27 (22, 30)	14 (13, 15)	14 (11, 18)
3	11 (10, 11)	49 (40, 56)	11 (10, 11)	13 (10, 19)

Table 13 Acceleration when using TMB for m state Poisson HMM parameter estimation, estimated on the hospital dataset. Each procedure was repeated and timed once on 100 bootstrap sample datasets. With 2 hidden states, *TMB_G* accelerated the estimation by an average factor of 27, with a 95% quantile bootstrap confidence interval of (22, 30)

m	<i>DM</i>	<i>TMB</i>	<i>TMB_G</i>	<i>TMB_H</i>	<i>TMB_{GH}</i>
2	2762.6 (2684.3, 3366)	1455.3 (1416.4, 1874.1)	2796.9 (2693.3, 3703.2)	1581.9 (1548.5, 1942.4)	481.9 (471.1, 501.1)
3	21562.9 (21053.3, 24389.2)	7108.2 (6967.6, 8349.3)	21445.4 (21051.3, 23429.3)	15396.4 (15185.4, 16571.1)	253.9 (248.1, 259.7)

Table 14 Duration (in millisecond) of m state Poisson HMM parameter estimation on the hospital dataset. Each procedure was timed 100 times. This table summarizes the computer's performance across multiple attempts at the same computation, in order to verify how reliable timing an estimation is. With 2 hidden states, estimation with *TMB_G* took an average 279 ms to compute, with an empirical 95% quantile confidence interval of 6.9 (2693.3)

All optimization methods

Finally, we compare different optimization methods. The ones we retain are BFGS, Nelder-Mead, L-BFGS-B, nlm, nlminb, and hjn, because the others don't converge in our case. `marqLevAlg` provides an algorithm for least-squares curve fitting, and is therefore included in the comparison. Exact gradients and Hessians are provided by TMB and fed to each algorithm. The speed comparisons are in the following Figure 2.

The following tables summarize the estimates and their confidence intervals, for the lamb dataset and for the simulated dataset. Instead of showing the standard error, we display the lower and upper bounds of the confidence intervals. This is due to profiling sometimes failing to provide both bounds of the interval.

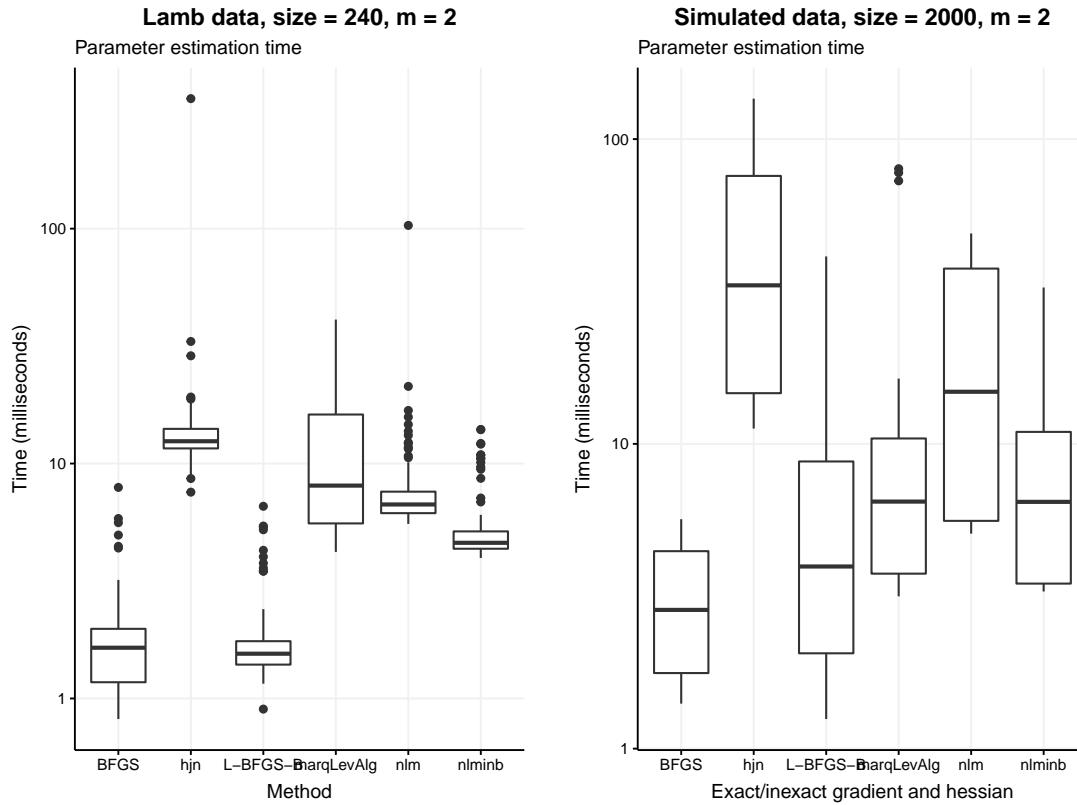


Figure 2 Poisson HMM parameter estimation time per optimization method

Parameter	Estimate	TMB CI		Profile CI		Bootstrap CI		Coverage prob.		
		L.	U.	L.	U.	L.	U.	TMB	Profile	Bootstrap
λ_1	0.26	0.15	0.33	0.00	0.33	0.18	0.34	0.80	0.90	0.90
λ_2	3.11	1.27	4.95	0.38	5.16	1.11	5.12	0.83	0.84	0.81
$\gamma_{1,1}$	0.99	0.93	1.00	0.54	1.00	0.97	1.01	0.97	0.97	0.96
$\gamma_{2,1}$	0.31	0.04	0.68	0.05	1.00	-0.05	0.67	0.96	1.00	0.84
$\gamma_{1,2}$	0.01	0.00	0.07	0.00	0.46	-0.01	0.03	0.97	0.97	0.96
$\gamma_{2,2}$	0.69	0.32	0.96	0.00	0.95	0.33	1.05	0.96	1.00	0.84
δ_1	0.96			0.31	0.99	0.90	1.03		1.00	0.88
δ_2	0.04			0.01	0.69	-0.03	0.10		1.00	0.88

Table 15 CIs for the lamb dataset. From left to right, the columns contain: the parameter name, parameter estimate, and lower (L.) and upper (U.) bound of the corresponding 95% CI derived by TMB, likelihood profiling, and percentile bootstrap. Then follow coverage probabilities derived for these three methods in a Monte-Carlo study.

m	Parameter	Value	Estimate	TMB CI		Profile CI		Bootstrap CI		Coverage prob.		
				L.	U.	L.	U.	L.	U.	TMB	Profile	Bootstrap
2	λ_1	1.00	0.97	0.91	1.03	0.78	1.14	0.91	1.03	0.95	0.94	0.95
2	λ_2	20.00	20.07	19.79	20.35	19.32	20.92	19.79	20.34	0.95	0.96	0.95
2	$\gamma_{1,1}$	0.80	0.80	0.77	0.82	0.71	0.86	0.77	0.82	0.93	0.92	0.92
2	$\gamma_{2,1}$	0.20	0.20	0.18	0.23	0.13	0.28	0.18	0.22	0.95	0.94	0.96
2	$\gamma_{1,2}$	0.20	0.20	0.18	0.23	0.14	0.29	0.18	0.23	0.93	0.92	0.92
2	$\gamma_{2,2}$	0.80	0.80	0.77	0.82	0.72	0.87	0.78	0.82	0.95	0.94	0.96
2	δ_1	0.50	0.50			0.36	0.62	0.45	0.54		0.92	0.93
2	δ_2	0.50	0.50			0.38	0.64	0.46	0.55		0.92	0.93
3	λ_1	1.00	0.95	0.87	1.02	0.71	1.18	0.87	1.02	0.95	0.95	0.95
3	λ_2	10.50	10.87	10.60	11.14	10.03	11.79	10.60	11.14	0.94	0.94	0.93
3	λ_3	20.00	20.19	19.77	20.62	18.90	21.54	19.77	20.61	0.94	0.95	0.94
3	$\gamma_{1,1}$	0.80	0.82	0.78	0.86	0.71	0.89	0.79	0.85	0.99	0.92	0.95
3	$\gamma_{2,1}$	0.10	0.09	0.07	0.11	0.03	0.16	0.07	0.11	0.96	0.94	0.94
3	$\gamma_{3,1}$	0.10	0.09	0.07	0.11	0.03	0.18	0.06	0.11	0.93	0.94	0.92
3	$\gamma_{1,2}$	0.10	0.10	0.08	0.12	0.04	0.19	0.08	0.12	0.90	0.92	0.96
3	$\gamma_{2,2}$	0.80	0.84	0.80	0.88	0.74	0.91	0.82	0.87	1.00	0.97	0.88
3	$\gamma_{3,2}$	0.10	0.08	0.06	0.11	0.02	0.20	0.05	0.11	0.92	0.94	0.90
3	$\gamma_{1,3}$	0.10	0.08	0.06	0.10	0.02	0.15	0.05	0.10	0.97	0.97	1.00
3	$\gamma_{2,3}$	0.10	0.07	0.05	0.09	0.02	0.15	0.05	0.09	0.94	0.94	0.89
3	$\gamma_{3,3}$	0.80	0.83	0.78	0.87	0.69	0.90	0.79	0.86	0.99	0.91	0.76
3	δ_1	0.33	0.33			0.19	0.49	0.28	0.39		0.94	0.94
3	δ_2	0.33	0.37			0.22	0.54	0.31	0.43		0.93	0.90
3	δ_3	0.33	0.29			0.15	0.44	0.24	0.35		0.94	0.93

Table 16 CIs for the simulated dataset. From left to right, the columns contain: the number of hidden states, parameter name, true parameter value, parameter estimate, and lower (L.) and upper (U.) bound of the corresponding 95% CI derived by TMB, likelihood profiling, and percentile bootstrap. Then follow coverage probabilities derived for these three methods in a Monte-Carlo study.

m	Parameter	Estimate	TMB CI		Profile CI		Bootstrap CI	
			L.	U.	L.	U.	L.	U.
2	λ_1	4.87	4.84	4.90	4.41	5.33	4.84	4.90
2	λ_2	13.33	13.29	13.37	12.72	14.04	13.29	13.37
2	$\gamma_{1,1}$	0.87	0.88	0.87	0.77	0.93	0.87	0.88
2	$\gamma_{2,1}$	0.10	0.10	0.10	0.05	0.17	0.10	0.10
2	$\gamma_{1,2}$	0.13	0.12	0.13	0.07	0.23	0.12	0.13
2	$\gamma_{2,2}$	0.90	0.90	0.90	0.83	0.95	0.90	0.90
2	δ_1	0.44			0.28	0.61	0.43	0.45
2	δ_2	0.56			0.39	0.72	0.55	0.57
3	λ_1	4.06	4.03	4.10	3.52	4.68	4.03	4.10
3	λ_2	10.47	10.40	10.54	9.55	11.22	10.40	10.54
3	λ_3	17.64	17.49	17.79	15.09	19.50	17.49	17.79
3	$\gamma_{1,1}$	0.85			0.71	0.92	0.85	0.86
3	$\gamma_{2,1}$	0.10			0.04	0.18	0.10	0.10
3	$\gamma_{3,1}$	0.00			0.00	0.10	-0.00	0.00
3	$\gamma_{1,2}$	0.10			0.01	0.22	0.10	0.11
3	$\gamma_{2,2}$	0.87			0.76	0.93	0.86	0.87
3	$\gamma_{3,2}$	0.17			0.05	0.39	0.16	0.18
3	$\gamma_{1,3}$	0.05			0.00	0.14	0.04	0.05
3	$\gamma_{2,3}$	0.03			0.00	0.10	0.03	0.03
3	$\gamma_{3,3}$	0.83			0.59	0.93	0.82	0.84
3	δ_1	0.33			0.16	0.49	0.32	0.34
3	δ_2	0.49			0.32	0.64	0.48	0.50
3	δ_3	0.18			0.07	0.35	0.17	0.19

Table 17 CIs for the hospital dataset. From left to right, the columns contain: the number of hidden states, parameter name, parameter estimate, and lower (L.) and upper (U.) bound of the corresponding 95% CI derived by TMB, likelihood profiling, and percentile bootstrap.

References

- Leonard E. Baum and Ted Petrie. Statistical Inference for Probabilistic Functions of Finite State Markov Chains. *The Annals of Mathematical Statistics*, 37(6):1554–1563, December 1966. ISSN 0003-4851. doi: 10.1214/aoms/1177699147.
- Leonard E. Baum, Ted Petrie, George Soules, and Norman Weiss. A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains. *The Annals of Mathematical Statistics*, 41(1):164–171, 1970. ISSN 00034851.
- Jan Bulla and Andreas Berzel. Computational issues in parameter estimation for stationary hidden Markov models. *Computational Statistics*, 23(1):1–18, January 2008. ISSN 1613-9658. doi: 10.1007/s00180-007-0063-y.
- Olivier Cappé, Eric Moulines, and Tobias Ryden. *Inference in Hidden Markov Models*. Springer Science & Business Media, April 2006. ISBN 978-0-387-28982-3.
- George Casella and Roger L. Berger. *Statistical Inference*. Cengage Learning, January 2021. ISBN 978-0-357-75313-2.
- Gilles Celeux and Jean-Baptiste Durand. Selecting hidden Markov model state number with cross-validated likelihood. *Computational Statistics*, 23(4):541–564, October 2008. ISSN 1613-9658. doi: 10.1007/s00180-007-0097-1.
- Mathias Drton and Martyn Plummer. A Bayesian information criterion for singular models. *arXiv:1309.0911 [stat]*, March 2016.
- Bradley Efron and Robert J Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall, New York, N.Y.; London, 1993. ISBN 978-0-412-04231-7.
- William Feller. *An Introduction to Probability Theory and Its Applications*. Wiley, 1968. ISBN 978-0-471-25708-0.
- Donald R. Fredkin and John A. Rice. Bayesian Restoration of Single-Channel Patch Clamp Recordings. *Biometrics*, 48(2):427–448, 1992. ISSN 0006341X, 15410420. doi: 10.2307/2532301.
- Sylvia Frühwirth-Schnatter. *Finite Mixture and Markov Switching Models*. Springer Science & Business Media, November 2006. ISBN 978-0-387-35768-3.
- Mark Gales and Steve Young. The Application of Hidden Markov Models in Speech Recognition. *Foundations and Trends® in Signal Processing*, 1(3):195–304, February 2008. ISSN 1932-8346, 1932-8354. doi: 10.1561/2000000004.
- Geoffrey Grimmett, Geoffrey R. Grimmett, Professor of Mathematical Statistics Geoffrey Grimmett, David Stirzaker, and Mathematical Institute David R. Stirzaker. *Probability and Random Processes*. OUP Oxford, May 2001. ISBN 978-0-19-857222-0.
- Wolfgang Härdle, Joel Horowitz, and Jens-Peter Kreiss. Bootstrap Methods for Time Series. *International Statistical Review*, 71(2):435–459, 2003. ISSN 1751-5823. doi: 10.1111/j.1751-5823.2003.tb00485.x.
- Robert E. Kass and Duane Steffey. Approximate Bayesian Inference in Conditionally Independent Hierarchical Models (Parametric Empirical Bayes Models). *Journal of the American Statistical Association*, 84(407):717–726, September 1989. ISSN 0162-1459. doi: 10.1080/01621459.1989.10478825.
- Kasper Kristensen, Anders Nielsen, Casper W. Berg, Hans Skaug, and Brad Bell. TMB: Automatic differentiation and Laplace approximation. *arXiv preprint arXiv:1509.00660*, 2015.
- Brian G. Leroux and Martin L. Puterman. Maximum-Penalized-Likelihood Estimation for Independent and Markov-Dependent Mixture Models. *Biometrics*, 48(2):545–558, 1992. ISSN 0006-341X. doi: 10.2307/2532308.
- Theodore C Lystig and James P Hughes. Exact Computation of the Observed Information Matrix for Hidden Markov Models. *Journal of Computational and Graphical Statistics*, 11(3):678–689, September 2002. ISSN 1061-8600. doi: 10.1198/106186002402.
- Geoffrey J. McLachlan and David Peel. *Finite Mixture Models*. John Wiley & Sons, March 2004. ISBN 978-0-471-65406-3.
- William Q. Meeker and Luis A. Escobar. Teaching about Approximate Confidence Regions Based on Maximum Likelihood Estimation. *The American Statistician*, 49(1):48–53, February 1995. ISSN 0003-1305. doi: 10.1080/00031305.1995.10476112.
- Jennifer Pohle, Roland Langrock, Floris van Beest, and Niels Martin Schmidt. Selecting the Number of States in Hidden Markov Models - Pitfalls, Practical Challenges and Pragmatic Solutions. *arXiv:1701.08673 [q-bio, stat]*, April 2017a.
- Jennifer Pohle, Roland Langrock, Floris M. van Beest, and Niels Martin Schmidt. Selecting the Number of States in Hidden Markov Models: Pragmatic Solutions Illustrated Using Animal Movement. *Journal of Agricultural*,

- Biological and Environmental Statistics*, 22(3):270–293, September 2017b. ISSN 1537-2693. doi: 10.1007/s13253-017-0283-8.
- Thomas Probst, Rüdiger Pryss, Berthold Langguth, and Winfried Schlee. Emotion dynamics and tinnitus: Daily life data from the “TrackYourTinnitus” application. *Scientific Reports*, 6(1):31166, August 2016. ISSN 2045-2322. doi: 10.1038/srep31166.
- R. Pryss, M. Reichert, J. Herrmann, B. Langguth, and W. Schlee. Mobile Crowd Sensing in Clinical and Psychological Trials – A Case Study. In *2015 IEEE 28th International Symposium on Computer-Based Medical Systems*, pages 23–24, June 2015a. doi: 10.1109/CBMS.2015.26.
- R. Pryss, M. Reichert, B. Langguth, and W. Schlee. Mobile Crowd Sensing Services for Tinnitus Assessment, Therapy, and Research. In *2015 IEEE International Conference on Mobile Services*, pages 352–359, June 2015b. doi: 10.1109/MobServ.2015.55.
- Eric E. Schadt, Janet S. Sinsheimer, and Kenneth Lange. Computational Advances in Maximum Likelihood Methods for Molecular Phylogeny. *Genome Research*, 8(3):222–233, January 1998. ISSN 1088-9051, 1549-5469. doi: 10.1101/gr.8.3.222.
- Rolf Turner. Direct maximization of the likelihood of a hidden Markov model. *Computational Statistics & Data Analysis*, 52(9):4147–4160, May 2008. ISSN 0167-9473. doi: 10.1016/j.csda.2008.01.029.
- D. J. Venzon and S. H. Moolgavkar. A Method for Computing Profile-Likelihood-Based Confidence Intervals. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 37(1):87–94, 1988. ISSN 1467-9876. doi: 10.2307/2347496.
- Ingmar Visser, Maartje E. J. Raijmakers, and Peter C. M. Molenaar. Confidence intervals for hidden Markov model parameters. *British Journal of Mathematical and Statistical Psychology*, 53(2):317–327, 2000. ISSN 2044-8317. doi: 10.1348/000711000159240.
- Abraham Wald. Tests of Statistical Hypotheses Concerning Several Parameters When the Number of Observations is Large. *Transactions of the American Mathematical Society*, 54(3):426–482, 1943. ISSN 0002-9947. doi: 10.2307/1990256.
- W. Zucchini, I.L. MacDonald, and R. Langrock. *Hidden Markov Models for Time Series: An Introduction Using R, Second Edition*. Chapman & Hall/CRC Monographs on Statistics & Applied Probability. CRC Press, 2016. ISBN 978-1-4822-5384-9.