Biometrical Journal **zz** (2021) zz, zzz–zzz / DOI: 10.1002/bimj.200100000

A gentle tutorial of accelerated parameter and confidence interval estimation for Hidden Markov Models using Template Model Builder

Timothée Bacri [1], **Geir D. Berentsen** [2], and **Jan Bulla**[*1]

[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

NOTE: the historical overview (1st par.) could also be moved to the very beginning of the introduction. Check in the end!
ML: do we need this abbrev.? How often?

Since their first application in speech recognition (see e.g. Baum and Petrie, 1966; Fredkin and Rice, 1992; Gales and Young, 2008), Hidden Markov Models (HMMs) found wide usage in many applied sciences. To name only a few, biology and bioinformatics (Schadt et al., 1998; Durbin, 1998; Eddy, 1998), finance (Hamilton, 1989), ecology (McClintock et al., 2020), stochastic weather modeling (Lystig and Hughes, 2002; Ailliot et al., 2015) and engineering (Mor et al., 2021).
A very common way to estimate the parameters of an HMM is the relatively straightforward computation of Maximum Likelihood (ML) estimates. For this task, most users rely on user-friendly implementation of the estimation routines via an interpreted programming language such as the statistical software environment `R` (R Core Team, 2021). Such an approach can easily require time-consuming computations, in particular for longer sequences of observations. In addition, selecting a suitable approach for deriving confidence intervals for the estimated parameters is not entirely obvious (see e.g. Zucchini et al., 2016; Lystig and Hughes, 2002; Visser et al., 2000), and often the computationally intensive bootstrap methods have to be applied.
In this tutorial, we illustrate how to speed up the computation of ML estimates significantly via the `R` package `TMB`. Moreover, this approach permits simple retrieval of standard errors at the same time. We illustrate the performance of our routines using different data sets. First, two smaller samples from a mobile application for tinnitus patients and a well-known data set of fetal lamb movements with 87 and 240 data points, respectively. Second, we rely on larger data sets of simulated data of sizes 2000 and 5000 for further analysis.
This tutorial is accompanied by a collection of scripts which are all available on GitHub. These scripts allow any user with moderate programming experience to benefit quickly from the computational advantages of `TMB`.

Key words: Hidden Markov Model; TMB; Confidence intervals; Maximum Likelihood Estimation; Tutorial

Supporting Information for this article is available from the author or on the WWW under `https://timothee-bacri.github.io/HMM_with_TMB`

*Corresponding author: e-mail: jan.bulla@uib.no, Phone: +47 55 58 28 75

© 2021 WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

1 Introduction

Hidden Markov models (HMMs) are a well-established, versatile type of model employed in many different applications. The scientific literature on this topic in statistics is rich, as illustrated e.g. by the manuscripts of Zucchini et al. (2016); Cappé et al. (2006); Bartolucci et al. (2012). The aforementioned sources contain, among many other aspects, detailed descriptions of parameter estimation for HMMs by maximization of the (log-)likelihood function. In short, Maximum likelihood estimation (MLE) is commonly achieved either by a direct numerical maximization as introduced by Turner (2008) and later detailed by Zucchini et al. (2016), who also provided a collection of R (R Core Team, 2021) scripts that is widely used. Alternatively, Expectation Maximization (EM) type algorithms as firstly described by Baum et al. (1970) or Dempster et al. (1977) serve for parameter estimation equally well. For a comparison of both approaches, see Bulla and Berzel (2008), who also describe a hybrid approach combining both algorithms. Evaluating uncertainty and obtaining confidence intervals (CIs) constitutes another essential aspect when working with HMMs - and it is less straightforward than parameter estimation. Although Cappé et al. (2006, Ch. 12) showed that CIs could be obtained based on 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) presents an alternative 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". In addition, Visser et al. (2000) report that computational problems arise when deriving CIs from the Hessian for sequences longer than 100 observations.

In this tutorial, we illustrate how to accelerate parameter estimation for HMMs with the help of Template Model Builder (TMB). As described by Kristensen et al. (2016), TMB is an R package developed for efficiently fitting complex statistical models to data. It provides exact calculations of first and second-order derivatives of the (log-)likelihood of a model by automatic differentiation, which allows for efficient gradient- and/or Hessian-based optimization of the likelihood on the one hand. On the other hand, TMB permits to infer CIs for the estimated parameters by means of the Hessian. We show how to carry out this part for HMMs using a couple of simple examples. Then, we compare the Hessian-based CIs with CIs resulting from likelihood profiling and bootstrapping, which are both more computationally intensive.

The tutorial is accompanied by a collection of scripts, which guide any user working with HMMs through the implementation of computationally efficient parameter estimation. The majority of scripts require only knowledge of R, just the computation of the (log-)likelihood function requires the involvement of C++. Moreover, we illustrate how TMB permits Hessian- or profile likelihood-based CIs for the estimated parameters at a very low computational cost. Naturally, the accelerated parameter estimation procedure may also serve for implementing computationally more efficient bootstrap CIs, an aspect we also make use of for our analyses.

Acknowledgements The work of J. Bulla was partially supported by the GENDER-Net Co-Plus Fund (GNP-182). We thank the University of Regensburg and European School for Interdisciplinary Tinnitus Research (ESIT) for providing access to the TYT data. Special thanks go to J. Simões for data preparation, and W. Schlee and B. Langguth for helpful comments and suggestions.

Conflict of Interest

The authors have declared no conflict of interest.

© 2021 WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

2 Principles of using TMB for MLE

In order to keep this tutorial at acceptable length, all sections follow the same concept. That is, the reader is encouraged to consult the respective part of our GitHub repository in parallel to reading each section; it is available at https://timothee-bacri.github.io/HMM_with_TMB. This permits to copy-paste or download all the scripts presented in this tutorial for each section on the one hand. On the other hand, it allows for consistent maintenance of the code. 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 $\boldsymbol{x}$ and $\boldsymbol{y}$ 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
```

© 2021 WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

```
  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 parametrized 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)
```

© 2021 WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

```
# 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 parametrized:

```
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 reparametrized 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`.

© 2021 WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

3 Parameter estimation techniques for HMMs

In this section we recall basic concepts underlying parameter estimation for HMMs via direct numerical optization 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 $\{X_t : t = 1, \ldots, T\}$ and $\{C_t : t = 1, \ldots, T\}$ denote the observed and hidden process, respectively. For an m-state Poisson HMM, the conditional distributions with parameter λ_i are then specified through

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

where $i = 1, \ldots, m$. 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 $X^{(t)} = \{X_1, \ldots, X_t\}$ and $x^{(t)} = \{x_1, \ldots, x_t\}$ denote the history of the observed process X_t and the observations x_t from time zero up to time t. Moreover, let $\boldsymbol{\theta}$ 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 $\{x_1, \ldots, x_T\}$ can then be expressed as

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

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 likelhood function, the so-called initial distribution, is given by the stationary distribution $\boldsymbol{\delta}$ here. Alternatively, the initial distribution may be estimated freely, which requires minor changes to the likelihood function discussed in

© 2021 WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

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 $\boldsymbol{\alpha}_t$ by

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

for $t = 1, 2, \dots, T$. The name forward algorithm originates from the way of calculating $\boldsymbol{\alpha}_t$, i.e.

$$\begin{aligned}\boldsymbol{\alpha}_0 &= \boldsymbol{\delta}\mathbf{P}(x_1)\\ \boldsymbol{\alpha}_t &= \boldsymbol{\alpha}_{t-1}\boldsymbol{\Gamma}\mathbf{P}(x_t) \text{ for } t = 1, 2, \dots, T.\end{aligned}$$

After a pass through all observations, the likelihood results from

$$L(\boldsymbol{\theta}) = \boldsymbol{\alpha}_T\mathbf{1}'.$$

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 $\boldsymbol{\beta}_t$ for $t = 1, 2, \dots, T$ so that

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

The name backward algorithm results from the way of calculating $\boldsymbol{\beta}_t$, i.e.

$$\begin{aligned}\boldsymbol{\beta}_T &= \mathbf{1}'\\ \boldsymbol{\beta}_t &= \boldsymbol{\Gamma}\mathbf{P}(x_{t+1})\boldsymbol{\beta}_{t+1} \text{ for } t = T-1, T-2, \dots, 1.\end{aligned}$$

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

$$L(\boldsymbol{\theta}) = \boldsymbol{\delta}\boldsymbol{\beta}_1.$$

In general, parameter estimation bases on the forward algorithm. The backward algorithm is, however, still useful because the quantities $\boldsymbol{\alpha}_t$ and $\boldsymbol{\beta}_t$ together serve for a couple of interesting tasks. For example,

© 2021 WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

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 on the GitHub page.
Last, it is well-known that the execution of the forward (or backward) algorithm may quickly lead to underflow errors, because many 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 Reparametrization of the likelihood function

The representation of the likelihood and the algorithms shown above rely on the data and the set of parameters $\boldsymbol{\theta}$ 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 $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_m)$ must be non-negative.

(ii) In general, the parameters γ_{ij} of the TPM $\boldsymbol{\Gamma}$ have to be non-negative, and the rows of $\boldsymbol{\Gamma}$ 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 reparametrize the log-likelihood in terms of unconstrained "working" parameters $\{\mathbf{T}, \boldsymbol{\eta}\} = g^{-1}(\boldsymbol{\Gamma}, \boldsymbol{\lambda})$, as follows. A possible reparametrization of $\boldsymbol{\Gamma}$ 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 $\boldsymbol{\Gamma}$ 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 reparametrized in terms of $\lambda_i = \exp(\eta_i)$, and consequently the unconstrained working parameters are given by $\eta_i = \log(\lambda_i), i = 1, \ldots, m$. Estimates of the "natural" parameters $\{\boldsymbol{\Gamma}, \boldsymbol{\lambda}\}$ can then be obtained by maximizing the reparametrized likelihood with respect to $\{\mathbf{T}, \boldsymbol{\eta}\}$ and then transforming the estimated working parameters back to natural parameters via the above transformations, i.e. $\{\hat{\boldsymbol{\Gamma}}, \hat{\boldsymbol{\lambda}}\} = g(\hat{\mathbf{T}}, \hat{\boldsymbol{\eta}})$. 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 MLE of the parameters of HMMs can be carried out efficiently via TMB.

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

© 2021 WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

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 {\tt{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 ”Track Your Tinnitus” (TYT) mobile application, a detailed description of which is presented in Pryss et al. (2015a) and Pryss et al. (2015b). We analyze 87 successive days of the “arousal” variable recorded for a single individual. This variable is measured on a discrete scale, where higher values correspond to a higher degree of excitement and lower values to a more calm emotional state (for details, see Probst et al., 2016, 2017).

Loading the “arousal” variable can be achieved simply with

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

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

(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

© 2021 WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

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

Table 1 TYT data. Observations collected by the TYT app on 87 successive days (from left to right) for a single individual.

(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)
```

© 2021 WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

```
## [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
```

It is noteworthy that various alternatives to `nlminb` exist. Nevertheless, we focus on this established optimization routine because of its high speed of convergence.

(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 $\boldsymbol{\delta}$ and accompanying standard errors, although $\boldsymbol{\delta}$ is not estimated, but derived from $\boldsymbol{\Gamma}$. We provide further details on this aspect in Section 5.1.

© 2021 WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

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 gradient and Hessian 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. $\log(\lambda)$). 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
```

© 2021 WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

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. 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)
```

It is noteworthy that more complex constraints are possible as well. For example, to impose equality constraints (such as $\gamma_{11} = \gamma_{22}$), the corresponding factor level has to be identical for the concerned entries. We refer to our GitHub page [INSERT GITHUB LINK](here) for more details on this. Last, 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 $i = 1, ..., m$, $t = 1, ..., n$, given all observations. These probabilities can be obtained by

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

© 2021 WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

where $\hat{\boldsymbol{\theta}}$ 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

$$i_t^* = \underset{i_t \in \{1,\ldots,m\}}{\arg\max}\ \mathrm{P}(C_t = i_t | X^{(n)} = x^{(n)}).$$

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

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

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

$$\mathrm{P}(X_{n+h} = x | X^{(n)} = x^{(n)}) = \frac{\boldsymbol{\alpha}_n \boldsymbol{\Gamma}^h \mathbf{P}(x) \mathbf{1}'}{\boldsymbol{\alpha}_n \mathbf{1}'} = \boldsymbol{\varphi}_n \boldsymbol{\Gamma}^h \mathbf{P}(x) \mathbf{1}',$$

where $\boldsymbol{\varphi}_n = \boldsymbol{\alpha}_n / \boldsymbol{\alpha}_n \mathbf{1}'$.
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. GEIR (OK?) Figure 1 shows the TYT data together with the conditional mean values linked to the most probable state inferred by the smoothing probabilities.

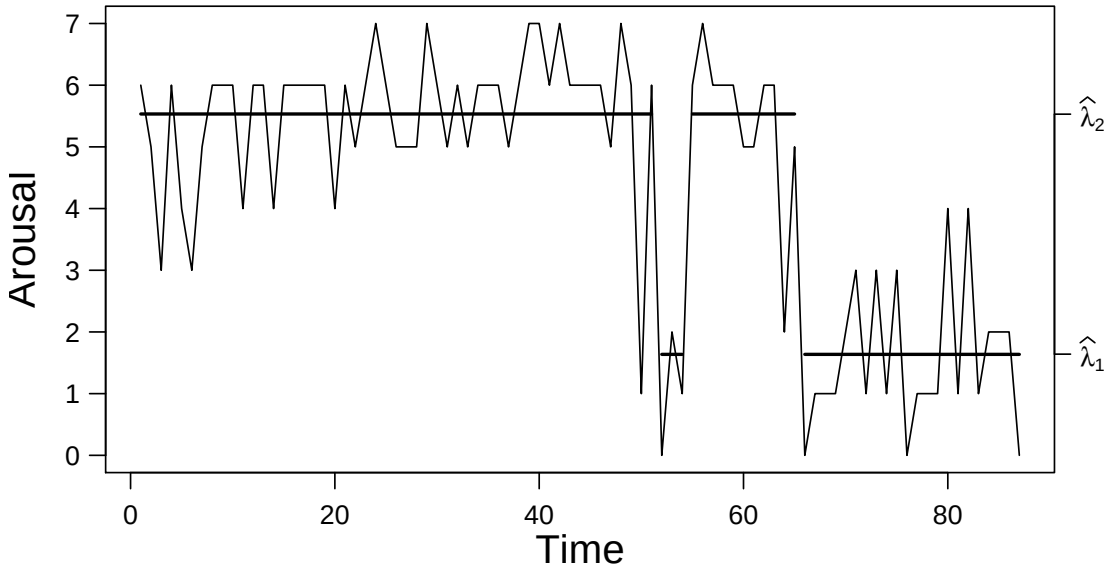

Figure 1 Plot of the TYT data. The solid horizontal lines correspond to the conditional mean of the inferred state at each time. See Table 5 for the values of $\widehat{\lambda}_i$.

© 2021 WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

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 $\mathbf{\Gamma}$ and $\boldsymbol{\lambda}$ and performance-related aspects in Section 6.

5.1 Wald-type confidence intervals based on the Hessian

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 $\{\hat{\mathbf{\Gamma}}, \hat{\boldsymbol{\lambda}}\}$ by

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

with $\{\hat{\mathbf{\Gamma}}, \hat{\boldsymbol{\lambda}}\} = g(\hat{\mathbf{T}}, \hat{\boldsymbol{\eta}})$ 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 $\mathbf{\Gamma}$ 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 (2017).

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),

© 2021 WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

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. $\mathbf{\Gamma}$, 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

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

In order to construct profile likelihood-based CIs, let $\{\hat{\mathbf{\Gamma}}, \hat{\boldsymbol{\lambda}}\}$ 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 $\log L(\{\hat{\mathbf{\Gamma}}, \hat{\boldsymbol{\delta}}\})$. 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:

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

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 $\hat{\mathbf{\Gamma}}, \hat{\boldsymbol{\lambda}}$. 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_1^2 distribution (e.g., 3.841 for $\alpha = 5\%$). The CI at level $1-\alpha$ for the parameter λ_2 is then given by

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

For simplicity, the principles of likelihood profiling shown above rely on the natural parameters. Our nll function is, however, parametrized 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 $\lambda_2 = \exp(\eta_2)$. For further details on the invariance of likelihood-based CIs to parameter transformations, we refer to Meeker and Escobar (1995).

`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)
par(mgp = c(2, 0.5, 0), mar = c(3, 3, 2.5, 1),
    cex.lab = 1.5)
plot(profile, level = 0.95,
     xlab = expression(eta[2]),
     ylab = "nll")
```

© 2021 WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

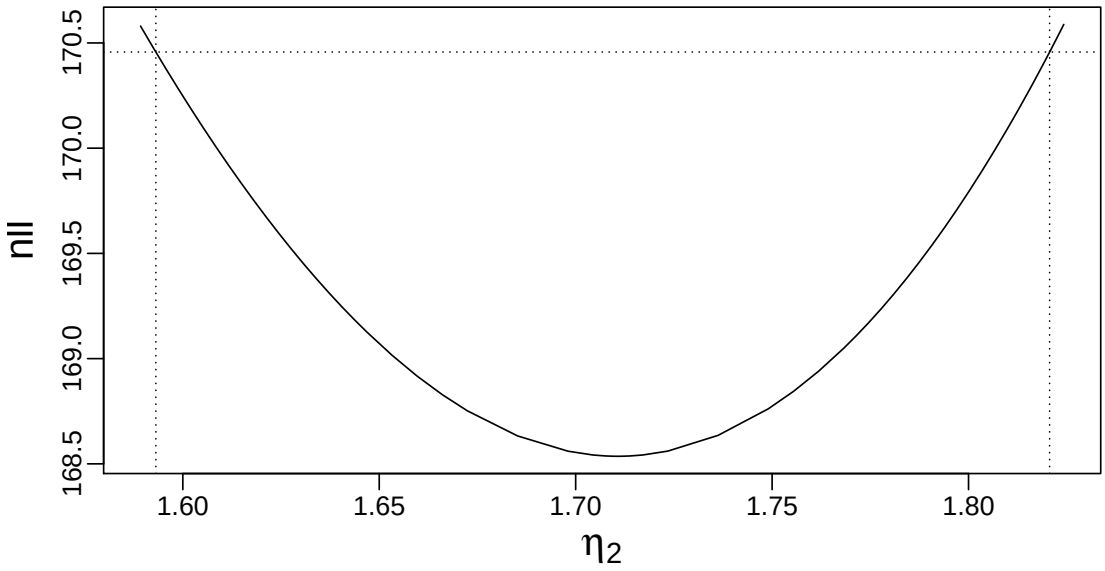

Figure 2 Profile likelihood plot. This figure shows the profile nll as function of the working parameter η_2. The vertical and horizontal lines correspond to the boundaries of the confidence interval and the critical value of the nll, respectively.

Furthermore, Figure 2 obtained via the `plot` function shows the resulting profile nll as function of the working parameter η_2. The vertical and horizontal lines correspond to the boundaries of the 95% CI 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
```

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 $\boldsymbol{\delta}$, 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 optmimization 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.

© 2021 WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

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 two-state Poisson HMM.

6 Application to different data sets

This section aims 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 simulated data sets. For the performance comparisons, the 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_0, TMB_H, TMB_G, and TMB_{GH} denote direct maximization with `TMB` without using the gradient and Hessian provided by `TMB`, with the Hessian, with the gradient, and with both gradient and Hessian, respectively.
As a preliminary reliability check of our IT infrastructure and setup, we timed the fitting of our two-state HMM to the TYT data with the help of the `microbenchmark` package (Mersmann, 2019). For this data set, all five approaches converged to the same optimum and parameter estimates, apart from minor variations typical for numerical optimization (see Table 2).

Par.	DM	TMB_0	TMB_G	TMB_H	TMB_{GH}
λ_1	1.636410931	1.636410932	1.636410933	1.636410932	1.636410997
λ_2	5.533095962	5.533095962	5.533095957	5.533095962	5.533095759
γ_{11}	0.949802041	0.949802041	0.949802042	0.949802041	0.949802094
γ_{12}	0.050197959	0.050197959	0.050197958	0.050197959	0.050197906
γ_{21}	0.025922044	0.025922044	0.025922044	0.025922044	0.025922038
γ_{22}	0.974077956	0.974077956	0.974077956	0.974077956	0.974077962
δ_1	0.340541816	0.340541816	0.340541819	0.340541816	0.340541999
δ_2	0.659458184	0.659458184	0.659458181	0.659458184	0.659458001
nll	168.536055869	168.536055869	168.536055869	168.536055869	168.536055869

Table 2 Parameter estimates and corresponding nll of the two-state Poisson HMM with and without using `TMB` obtained for the TYT data.

© 2021 WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

Table 3 shows the resulting average time required for the parameter estimation and the number of iterations needed by each approach, measured over 200 replications. The results show that the use of `TMB` significantly accelerates parameter estimation in comparison with DM. The most substantial acceleration is achieved by TMB_G, underlining the benefit of using the gradient provided by `TMB`. Moreover, TMB_{GH} requires fewer iterations than the other approaches. However, the evaluation of the Hessian seems to increase the computational burden.

	DM	TMB_0	TMB_G	TMB_H	TMB_{GH}
Time (ms)	23.1	1.71	0.804	1.67	2.18
	(22.5, 23.7)	(1.62, 1.8)	(0.788, 0.82)	(1.66, 1.68)	(2.11, 2.26)
Iterations	13	13	13	13	7

Table 3 Average duration (in milliseconds) together with 95% CI and number of iterations required for fitting a two-state Poisson HMM to the TYT data. The CIs are of Wald-type and base on the standard error of the mean derived from 200 replications.

Next, we verified the reproducibility of the acceleration by `TMB` in a parametric bootstrap setting. More specifically, we simulated 200 bootstrap 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 most substantial acceleration and TMB_{GH} reducing the number of iterations. This underlines that two factors are sources of the acceleration in Table 4: the use of `C++` code on the one hand, and computation of the gradient and/or Hessian by `TMB` on the other hand.

	TMB_0	TMB_G	TMB_H	TMB_{GH}
Acceleration ratio	12.9	26.5	12.8	11.3
	(11.8, 13.8)	(22.9, 29.5)	(11.5, 13.8)	(8.69, 14.3)
Iterations	14.5	14.5	14.5	7.1
	(11, 20)	(11, 20)	(11, 20)	(5, 11)

Table 4 Acceleration and iterations for the TYT data. The top lines show the acceleration ratios together with 95% percentile bootstrap CIs when using `TMB` in a bootstrap setting with 200 bootstrap samples. The bottom lines display the corresponding values for the number of iterations.

In order to obtain reliable results, we excluded all those bootstrap samples with a simulated state sequence not sojourning in each state of the underlying model at least once. This is necessary because such a constellation almost certainly leads to convergence problems on the one hand. On the other hand, even if the estimation algorithms converge, the estimated models are usually degenerate because of the lack of identifiability. Furthermore, for some very rare 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 mainly occurred due to TMB_0 and TMB_H failing. Therefore, we recommend passing at least the gradient when optimizing with `TMB` for increased stability.

Last, we investigate CIs obtained for the TYT data by the three different methods described in Section 5, TMB_{GH} served as the sole estimation approach. The columns to the left in Table 5 show the parameter estimates and the three types of 95% CIs obtained using the Hessian, likelihood profiling, and bootstrapping. For this data set, no major differences between the different CIs are visible. Furthermore, we assessed the accuracy of the different CIs by computing coverage probabilities, which are shown in

© 2021 WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

the last three columns of Table 5. For calculating these coverage probabilities, we used a Monte Carlo setting similar to the one described above. Samples that possessed state sequences not visiting all states or samples for which the estimation algorithm did not converge were replaced. Moreover, we also simulated a replacement sequence when the profile likelihood method failed to converge on any bound to ensure comparability of the results. The results, shown on the right of Table 5 indicate that all methods provide comparably reliable CI estimates, and neither outperforms the other for all parameters. For the Wald-type CIs, the coverage probabilities almost reach 100% for γ_{22}, γ_{21}, but lie comparably low for both the other elements of the TPM and δ_1, δ_2. However, profile likelihood-based CIs also take values above 95% for all elements of the TPM, and the coverage probabilities for bootstrap CIs are all above 95%.

		Wald-type CI		Profile CI		Bootstrap CI		Coverage prob. (%)		
Par.	Est.	L.	U.	L.	U.	L.	U.	Wald	Profile	Bootst.
λ_1	1.64	1.09	2.18	1.15	2.23	0.83	2.96	93.9	96.1	99.1
λ_2	5.53	4.91	6.16	4.92	6.18	4.73	6.43	93.8	94.4	97.7
γ_{11}	0.95	0.86	1.00	0.82	1.00	0.45	0.99	90.2	96.6	95.2
γ_{12}	0.05	0.00	0.14	0.00	0.18	0.01	0.55	90.2	96.6	95.2
γ_{21}	0.03	0.00	0.07	0.00	0.09	0.01	0.20	99.6	97.3	96.0
γ_{22}	0.97	0.93	1.00	0.91	1.00	0.80	0.99	99.6	97.3	96.0
δ_1	0.34	0.00	0.79			0.07	0.82	87.3		95.6
δ_2	0.66	0.21	1.00			0.18	0.93	87.3		95.6

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 via the Hessian provided by TMB, likelihood profiling, and percentile bootstrap. Then follow coverage probabilities derived for these three methods in a Monte-Carlo study.

TODO: VALUES BELOW FOR CONTROL. BENCHMARK SAMPLES IS INTENTIONALLY LOW

CONSISTENCY_BENCHMARK_TINN=200 replications for the first speed TYT benchmarking.
BENCHMARK_SAMPLES=200 replications for the speed benchmarking.
BOOTSTRAP_SAMPLES=1000 replications for bootstrap CIs
COVERAGE_SAMPLES=1000 replications for coverage CIs

6.2 Lamb data

We fit two-state HMMs to the well-known data set presented in Leroux and Puterman (1992) as the second example. This data set consists of the number of movements by a fetal lamb observed through ultrasound during 240 consecutive 5-second intervals, as shown in Figure 3.

We selected this data set for several reasons. First, the number of observations is larger than for the TYT data (but still comparably low). Secondly, according to the results of Leroux and Puterman (1992), the first state largely dominates the data generating process, whereas the second state is not very persistent and linked to only a few observations. Thirdly, the conditional means of the two states are not very different. The latter two aspects qualify this data as a 'non-text-book example'.

Similar to the TYT data, all estimations algorithms converged to the same minimum of the nll, and provided almost identical parameter estimates on the original data set. A bootstrap experiment similar to the one described above for the TYT data led to comparable results, as shown in Table 6. The highest acceleration is achieved by TMB_G, whereas TMB_{GH} achieves the lowest acceleration despite requiring a lower number of iterations than the other approaches. However, all ratios lie above those obtained for the TYT data, indicating an increased benefit of using TMB with an increasing number of observations.

Next, Table 7 shows parameter estimates and corresponding CIs in the columns to the left. Our estimates confirm the results of Leroux and Puterman (1992): the second state is not very persistent, and the

© 2021 WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

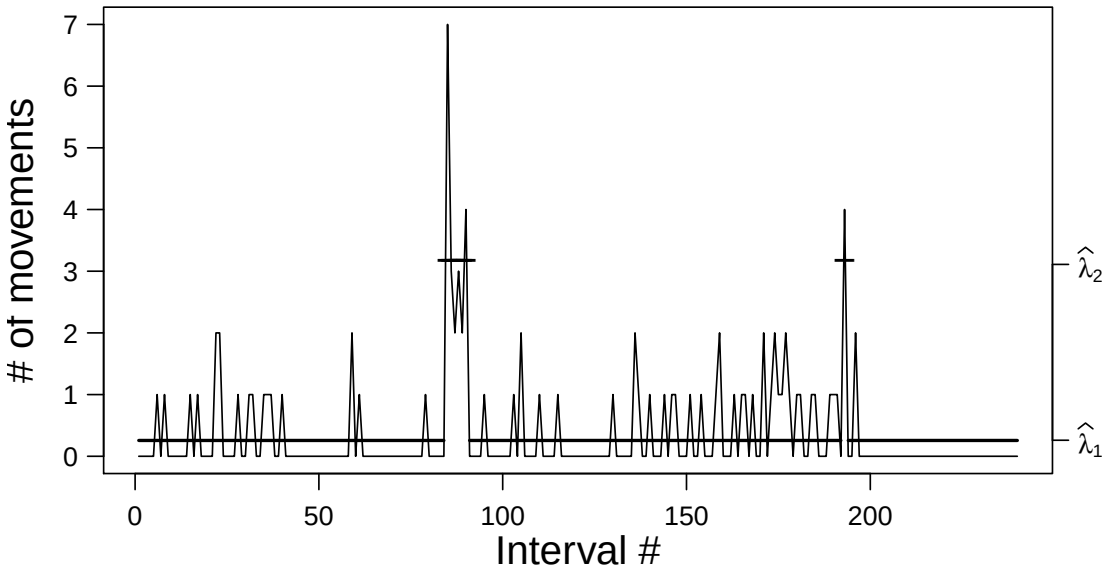

Figure 3 Plot of the lamb data. The solid horizontal lines correspond to the conditional mean of the inferred state at each time. See Table 7 for the values of $\widehat{\lambda}_i$.

	TMB_0	TMB_G	TMB_H	TMB_{GH}
Acceleration ratio	21.2	41.4	21.2	12
	(17.6, 23.6)	(36.6, 51.3)	(17.7, 23.5)	(8.95, 17.9)
Iterations	15.4	15.4	15.4	8.85
	(11, 38)	(11, 39)	(11, 38)	(6, 23)

Table 6 Acceleration and iterations for the lamb data. The top lines show the acceleration ratios together with 95% percentile bootstrap CIs when using `TMB` in a bootstrap setting with 200 bootstrap samples. The bottom lines display the corresponding values for the number of iterations.

conditional means λ_1 and λ_2 do not lie very far from each other. Concerning the CIs resulting from our three approaches, it is noticeable that the bootstrap CIs for the elements of the TPM are larger than those obtained by the other two approaches. The coverage probabilities presented in the columns to the right of Table 7 show similar patterns as observed for the TYT data. Wald-type CIs show certain deviations from 95% for the parameters related to the hidden state sequence, and the same is true for the profile CIs. Bootstrap CIs seem to be slightly too large for most parameters, leading to values greater than 95%.

6.3 Simulation study

The two previously analyzed data sets are both of comparably small size. In order to systematically investigate the performance of `TMB` in the context of larger samples, we carried out a small simulation study. For this study, we simulated sequences of observations of length 2000 and 5000 from HMMs with two and three states, respectively. The parameters underlying the simulation are

$$\mathbf{\Gamma} = \begin{pmatrix} 0.95 & 0.05 \\ 0.15 & 0.85 \end{pmatrix}, \quad \boldsymbol{\lambda} = (1, 7)$$

© 2021 WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

		Wald-type CI		Profile CI		Bootstrap CI		Coverage prob. (%)		
Par.	Est.	L.	U.	L.	U.	L.	U.	Wald	Profile	Bootst.
λ_1	0.26	0.18	0.34	0.15	0.33	0.11	0.33	93.8	94.6	95.3
λ_2	3.11	1.11	5.12	1.27	4.95	0.40	5.20	92.6	94.0	96.8
γ_{11}	0.99	0.97	1.00	0.93	1.00	0.69	1.00	99.9	95.3	97.0
γ_{12}	0.01	0.00	0.03	0.00	0.07	0.00	0.31	99.9	95.3	97.0
γ_{21}	0.31	0.00	0.67	0.04	0.68	0.08	1.00	93.6	97.3	99.3
γ_{22}	0.69	0.33	1.00	0.32	0.96	0.00	0.92	93.6	97.3	99.3
δ_1	0.96	0.90	1.00			0.45	0.99	98.5		98.2
δ_2	0.04	0.00	0.10			0.01	0.55	98.5		98.2

Table 7 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 via the Hessian provided by TMB, likelihood profiling, and percentile bootstrap. Then follow coverage probabilities derived for these three methods in a Monte-Carlo study.

for the two-state HMM and

$$\mathbf{\Gamma} = \begin{pmatrix} 0.95 & 0.025 & 0.025 \\ 0.05 & 0.90 & 0.05 \\ 0.075 & 0.075 & 0.85 \end{pmatrix}, \quad \boldsymbol{\lambda} = (1, 4, 7)$$

for the HMM with three states. Figure 4 and Figure 5 display the first 500 observations of two exemplary sequences of observations generated for the two- and three-state model, respectively.

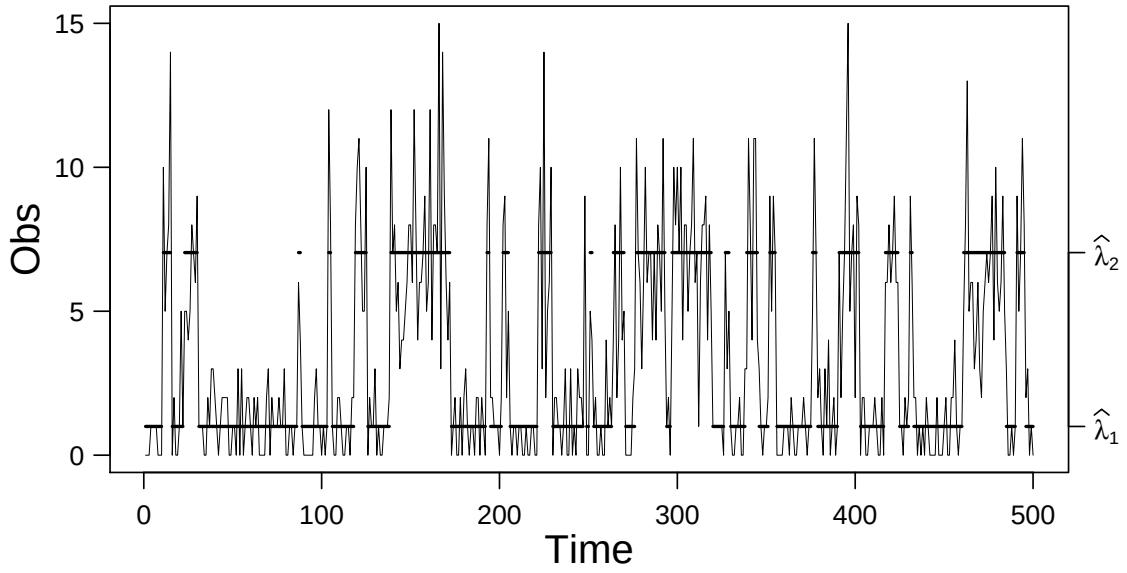

Figure 4 Plot of simulated data (of size 2000) generated by a two-state Poisson HMM. The solid horizontal lines correspond to the conditional mean of the inferred state at each time. For readability, the graph is truncated to 500 data points. See Table 10 for the values of $\widehat{\lambda}_i$.

© 2021 WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

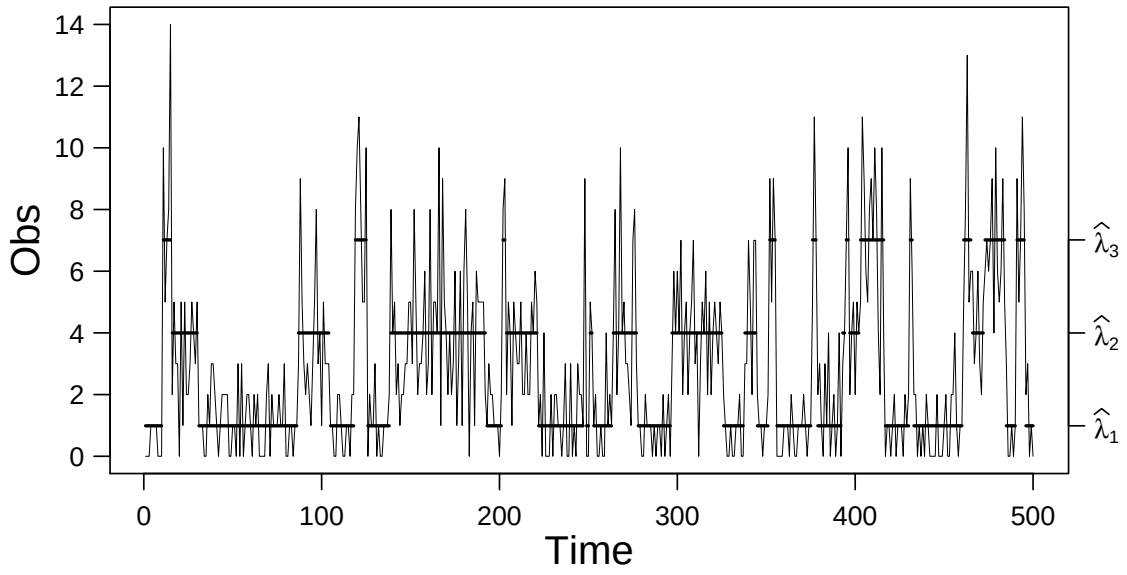

Figure 5 Plot of simulated data (of size 5000) generated by a three-state Poisson HMM. The solid horizontal lines correspond to the conditional mean of the inferred state at each time. For readability, the graph is truncated to 500 data points. See Table 11 for the values of $\widehat{\lambda}_i$.

The bootstrap setting described in the previous sections served for computing acceleration ratios for the simulated data sets. Table 8 displays the results obtained for the two-state model. Again, TMB_G provides the strongest acceleration compared to DM. Furthermore, TMB_{GH} achieves the second place, closely followed by the remaining two approaches. This confirms the good performance of TMB_G for large sample as well. Furthermore, it suggests a certain benefit from employing the Hessian computed by `TMB` for longer observation sequences, since this method requires a much lower number of iterations than all other approaches.

	TMB_0	TMB_G	TMB_H	TMB_{GH}
Acceleration ratio	18.1	36.2	18.1	19.7
	(16.6, 21.1)	(30.9, 45.2)	(16.4, 21)	(14.6, 25.8)
Iterations	20.7	20.6	20.7	5.82
	(17, 25)	(16, 26)	(17, 25)	(5, 8)

Table 8 Acceleration and iterations for the simulated data of size 2000. The top lines show the acceleration ratios together with 95% percentile bootstrap CIs when using `TMB` in a bootstrap setting with 200 bootstrap samples. The bottom lines display the corresponding values for the number of iterations.

The results for the three-state model, as shown in Table 9, basically underline all findings from the two-state model. Most importantly, again TMB_{GH} requires a much lower number of iterations than the other approaches.

Similar to the two previous sections, Table 10 and Table 11 show parameter estimates, CIs, and coverage probabilities. The two exemplary sequences of observations shown in Figure 4 and Figure 5 served for deriving parameter estimates and CIs. The coverage probabilities result from a Monte-Carlo study as the

© 2021 WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

	TMB_0	TMB_G	TMB_H	TMB_{GH}
Acceleration ratio	12.9	49.5	12.9	27.5
	(11.8, 14.3)	(42.4, 57.8)	(11.5, 14.3)	(20.7, 32.7)
Iterations	44.7	44.2	44.7	5.33
	(36.9, 54)	(36, 52.1)	(36.9, 54)	(5, 8)

Table 9 Acceleration and iterations for the simulated data of size 5000. The top lines show the acceleration ratios together with 95% percentile bootstrap CIs when using TMB in a bootstrap setting with 200 bootstrap samples. The bottom lines display the corresponding values for the number of iterations.

ones previously described. Overall, the CIs obtained by our three methods are very similar for all parameters. The coverage probabilities lie comparably close to the theoretical level of 95% for all parameters of the two-state model. Moreover, no systematically too small or large CIs seem to result from any of the three methods. The same holds true for the three-state model, with minor exceptions for the profile CIs. For this method, the coverage probabilities of the CIs are close to 100% for diagonal elements of the TPM, while the corresponding probabilities for off-diagonal elements are less than 95%.

			Wald-type CI		Profile CI		Bootstrap CI		Coverage prob. (%)		
Par.	Value	Est.	L.	U.	L.	U.	L.	U.	Wald	Profile	Bootst.
λ_1	1.00	1.04	0.99	1.10	0.99	1.10	0.99	1.10	95.3	95.1	95.4
λ_2	7.00	6.95	6.71	7.19	6.72	7.19	6.69	7.21	96.0	95.7	96.0
γ_{11}	0.95	0.95	0.94	0.96	0.94	0.96	0.94	0.96	95.8	95.5	95.3
γ_{12}	0.05	0.05	0.04	0.06	0.04	0.06	0.04	0.06	95.8	95.5	95.3
γ_{21}	0.15	0.14	0.11	0.18	0.11	0.18	0.11	0.18	93.8	94.5	94.5
γ_{22}	0.85	0.86	0.82	0.89	0.82	0.89	0.82	0.89	93.8	94.5	94.5
δ_1	0.75	0.74	0.68	0.80			0.68	0.80	94.3		94.1
δ_2	0.25	0.26	0.20	0.32			0.20	0.32	94.3		94.1

Table 10 CIs for the simulated dataset of size 2000. 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 via the Hessian provided by TMB, likelihood profiling, and percentile bootstrap. Then follow coverage probabilities derived for these three methods in a Monte-Carlo study.

© 2021 WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

			Wald-type CI		Profile CI		Bootstrap CI		Coverage prob. (%)		
Par.	Value	Est.	L.	U.	L.	U.	L.	U.	Wald	Profile	Bootst.
λ_1	1.000	1.000	0.96	1.04	0.96	1.04	0.96	1.04	93.6	93.6	93.5
λ_2	4.000	4.027	3.85	4.20	3.85	4.20	3.87	4.19	95.6	95.6	95.0
λ_3	7.000	6.906	6.66	7.15	6.67	7.16	6.65	7.18	94.4	94.8	95.4
γ_{11}	0.950	0.943	0.93	0.95	0.92	0.96	0.93	0.95	95.3	99.9	94.5
γ_{12}	0.025	0.029	0.02	0.04	0.02	0.04	0.02	0.04	94.2	93.4	93.8
γ_{13}	0.025	0.029	0.02	0.04	0.02	0.04	0.02	0.04	94.8	93.8	94.6
γ_{21}	0.050	0.039	0.02	0.06	0.02	0.05	0.02	0.06	94.7	93.1	94.7
γ_{22}	0.900	0.907	0.88	0.93	0.87	0.94	0.88	0.93	94.0	99.6	94.6
γ_{23}	0.050	0.054	0.03	0.08	0.03	0.08	0.03	0.08	93.3	93.8	94.6
γ_{31}	0.075	0.094	0.07	0.12	0.07	0.11	0.07	0.12	94.5	90.6	94.4
γ_{32}	0.075	0.055	0.03	0.08	0.03	0.08	0.03	0.09	93.0	90.8	93.4
γ_{33}	0.850	0.851	0.82	0.88	0.80	0.89	0.81	0.88	94.1	99.7	94.5
δ_1	0.545	0.520	0.46	0.58			0.46	0.58	94.8		95.2
δ_2	0.273	0.279	0.22	0.33			0.23	0.34	94.9		94.9
δ_3	0.182	0.201	0.16	0.24			0.16	0.24	94.3		94.4

Table 11 CIs for the simulated dataset of size 5000. 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 via the Hessian provided by `TMB`, likelihood profiling, and percentile bootstrap. Then follow coverage probabilities derived for these three methods in a Monte-Carlo study.

© 2021 WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

7 Discussion

In this tutorial, we provide researchers from all applied fields with an introduction to parameter estimation for HMMs via TMB using R. Although some procedures need to be coded in C++, the use of TMB permits to accelerate existing parameter estimation procedures without having to carry out major changes to R code which is already in use. Moreover, after finishing the estimation procedure, TMB permits to obtain standard errors for the estimated parameters at a very low computational cost.

We examined the performance of TMB in the context of Poisson HMMs through two small real data sets and in a simulation setting with longer sequences of observations. Overall, it is notable that the parameter estimation process is strongly accelerated on the one hand. This applies even to small data sets, and the highest acceleration is obtained when only the gradient is supplied by TMB (instead of both gradient and Hessian). On the other hand, the standard errors obtained through TMB are very similar to the standard errors obtained by profiling the likelihood and bootstrapping while being (much) less computationally intensive. This is novel since Hessian-base CIs did not seem to be reliable in the past, as illustrated e.g. by Visser et al. (2000).

Along with the tutorial character of this paper comes the shortcoming that we restricted ourselves to only one comparably simple HMM with Poisson conditional distributions. The extension to other distributions is, however, not overly complicated. We briefly illustrate the case of Gaussian conditional distributions on the supporting GitHub page. Moreover, to keep the paper at acceptable length, we were not able to address a couple of potential extensions and research questions. For example, it would be interesting to investigate whether supplying the Hessian provided by TMB to the optimizer has a positive impact on convergence properties, in particular, if the initial values are poor. Then, one could check to which extent other optimizing functions (e.g. `nlm`) benefit from employing TMB. The reliability of Wald-type CIs provided by TMB for other models and very long sequences with e.g. hundreds of thousands or millions of observations could also be of interest. In a similar direction, it seems rather obvious to benefit from TMB for more complex settings such as panel data with random effects, where computationally efficient routines play an important role. Last, in the bootstrap analysis of the TYT and in particular of the lamb data set, we noted that the TMB-internal function `tmbprofile` sometimes fails to provide CIs for parameters very close to a boundary. For these two data sets, this was the case for elements of the TPM close to one, and roughly 7% (TYT) and 22% (lamb), respectively, of the generated data sets were affected. It remains to clarify whether this problem can be solved by a suitable modification of `tmbprofile`, or if the underlying difficulties require an entirely different approach.

From an application perspective, the use of TMB allows executing estimation procedures at a significantly reduced cost compared to the execution of plain R. Such a performance gain could be of interest when repeatedly executing statistical procedures on mobile devices. It seems plausible to enrich the TYT app (or similar apps collecting a sufficient amount of data) by integrating a warning system that detects when the user enters a new state, inferred through a suitable HMM or another procedure accelerated via TMB in real-time. This new state could, e.g., represent an improvement or worsening of a pre-defined medical condition and recommend the user to contact the consulting physician if the change persists for a certain period. Provided the agreement of the user, this collected information could also be pre-processed and transferred automatically to the treating physician and allow to identify personalized treatment options.

© 2021 WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

References

Pierre Ailliot, Denis Allard, Valérie Monbet, and Philippe Naveau. Stochastic weather generators: an overview of weather type models. *Journal de la société française de statistique*, 156(1):101–113, 2015.

F. Bartolucci, A. Farcomeni, and F. Pennoni. *Latent Markov Models for Longitudinal Data*. Chapman & Hall/CRC Statistics in the Social and Behavioral Sciences. Taylor & Francis, 2012. ISBN 978-1-4398-1708-7.

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.

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.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data Via the EM Algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977. ISSN 2517-6161. doi: 10.1111/j.2517-6161.1977.tb01600.x.

Mathias Drton and Martyn Plummer. A Bayesian information criterion for singular models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 79(2):323–380, 2017. ISSN 1467-9868. doi: 10.1111/rssb.12187.

Richard Durbin. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. April 1998.

S R Eddy. Profile hidden Markov models. *Bioinformatics*, 14(9):755–763, January 1998. ISSN 1367-4803. doi: 10.1093/bioinformatics/14.9.755.

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 Series in Statistics. Springer-Verlag, New York, 2006. ISBN 978-0-387-32909-3. doi: 10.1007/978-0-387-35768-3.

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.

James D. Hamilton. A New Approach to the Economic Analysis of Nonstationary Time Series and the Business Cycle. *Econometrica*, 57(2):357–384, 1989. ISSN 0012-9682. doi: 10.2307/1912559.

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 Bradley M. Bell. TMB: Automatic differentiation and Laplace approximation. *Journal of Statistical Software*, 70(5):1–21, 2016. doi: 10.18637/jss.v070.i05.

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.

© 2021 WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

doi: 10.1198/106186002402.

Brett T. McClintock, Roland Langrock, Olivier Gimenez, Emmanuelle Cam, David L. Borchers, Richard Glennie, and Toby A. Patterson. Uncovering ecological state dynamics with hidden Markov models. *Ecology Letters*, 23(12): 1878–1903, 2020. ISSN 1461-0248. doi: 10.1111/ele.13610.

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.

Olaf Mersmann. *Microbenchmark: Accurate Timing Functions*, 2019.

Bhavya Mor, Sunita Garhwal, and Ajay Kumar. A Systematic Review of Hidden Markov Models and Their Applications. *Archives of Computational Methods in Engineering*, 28(3):1429–1448, May 2021. ISSN 1886-1784. doi: 10.1007/s11831-020-09422-4.

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.

Thomas Probst, Rüdiger C. Pryss, Berthold Langguth, Josef P. Rauschecker, Johannes Schobel, Manfred Reichert, Myra Spiliopoulou, Winfried Schlee, and Johannes Zimmermann. Does Tinnitus Depend on Time-of-Day? An Ecological Momentary Assessment Study with the "TrackYourTinnitus" Application. *Frontiers in Aging Neuroscience*, 9:253, August 2017. ISSN 1663-4365. doi: 10.3389/fnagi.2017.00253.

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.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2021.

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.

© 2021 WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim