Biometrical Journal **52** (2020) 61, zzz–zzz / DOI: 10.1002/bimj.200100000

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

## Timothee Bacri∗ 1, Jan Bulla 1, and Geir D. Berentsen 2

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](#_bookmark27) [and Young](#_bookmark27), [2008](#_bookmark27); [Fredkin and Rice](#_bookmark25), [1992](#_bookmark25)). There are straightforward ways to compute Maximum Likeli- hood Estimates (MLE) of their parameters, but getting confidence intervals can be more difficult (See e.g. [Zucchini et al.](#_bookmark40), [2016](#_bookmark40); [Lystig and Hughes](#_bookmark33), [2002](#_bookmark33)). In addition, computing MLEs can be time-consuming when the dataset and the complexity become very large. We show in this paper a way to speed up com- putation by up to 47 times in the language R when compared to usual optimization approaches, and at the same time retrieve standard errors easily. Here, we show in a first part how to optimize HMMs with the TMB package in R and how to retrieve confidence intervals. In a second part, we compare different optimiz- ers such as nlminb, and minimize the negative log-likelihood directly on different datasets: a small one (240 data points) from [Leroux and Puterman](#_bookmark32) ([1992](#_bookmark32)), a medium sized simulated one (2000), and a large one (87648) 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

Hidden Markov models (HMM) are a versatile type of model that have used in many different situations since their introduction by [Baum and Petrie](#_bookmark21) ([1966](#_bookmark21)). As an example, [Fredkin and Rice](#_bookmark25) ([1992](#_bookmark25)) applied them in speech recognition, [Lystig and Hughes](#_bookmark33) ([2002](#_bookmark33)) to rainfall occurence data, and [Schadt et al.](#_bookmark36) ([1998](#_bookmark36)) to phylogenetetic trees. [Fredkin and Rice](#_bookmark25) ([1992](#_bookmark25)) and [Zucchini et al.](#_bookmark40) ([2016](#_bookmark40)) are references in the theory of HMMs. Evaluating uncertainty and getting confidence intervals in HMMs is not easy. Although [Cappe´](#_bookmark22) [et al.](#_bookmark22) ([2006](#_bookmark22), Chapter 12) showed it can be achieved using asymptotic normality of the MLEs of the param- eters under certain conditions, [Fru¨hwirth-Schnatter](#_bookmark26) ([2006](#_bookmark26)) points out that in indepedent mixture models, “The regularity conditions are often violated”, and [McLachlan and Peel](#_bookmark34) ([2004](#_bookmark34), p. 68) adds that “In partic- ular 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](#_bookmark33) ([2002](#_bookmark33)) shows a way to compute the exact Hessian, and [Zucchini et al.](#_bookmark40) ([2016](#_bookmark40)) shows another way to compute the approximate Hessian and thus con- fidence 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”.

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.](#_bookmark31) ([2015](#_bookmark31)). It provides exact calculations of first and second

∗Corresponding author: e-mail: [timothee.bacri@uib.no,](mailto:timothee.bacri@uib.no) Phone: +33-636-775-063

2 First Author*et al.* and dd: Running title

order derivatives of the likelihood of a model by automatic differentiation, which allows for efficient gra- dient 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 adressed by constraint optimiza- tion, and subsequently combining the Hessian with the Jacobian of the constraints to obtain the covariance matrix as shown by [Visser et al.](#_bookmark38) ([2000](#_bookmark38)). Alternatively, [Zucchini et al.](#_bookmark40) ([2016](#_bookmark40)) shows how the constraints can be imposed by suitable transformations of the parameters. The covariance matrix of the untransformed (original) parameters can then be obtained by the delta method, a feature implemented in TMB.

In this paper, we will show how to optimize a Poisson HMM using TMB in the language R. Afterwards, we will use an example to show how to make a nested model, easily compute confidence intervals, retrieve interesting probabilities, we will apply a Poisson HMM on a hospital dataset. Eventually, we will see that TMB can accelerate traditional optimizers in R by up to a few hundred times on a large dataset, simplifies evaluating confidence intervals, and gives similar parameter uncertainty to bootstrap and profile methods via its Hessian based approach.

We decided to use a direct maximization approach instead of the Expectation-Maximization (EM) algo- rithm. 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.

# Principles of using TMB for Maximum Likelihood Estimation (MLE)

## Setup

In order to use TMB in R, it is necessary to use the R-package TMB and have the software Rtools

installed. The latter is used for compiling the C++ code.

In this paper, we will use R version 3.6.0 and the code was run on Windows 10 Enterprise version 1809, with an Intel(R) Core(TM) i7-8700 processor.

The C++ code can be difficult to debug as it operates using a specific template, see [Section 2.2](#_bookmark1) below for an example. TMB provides a debugging feature available with RStudio which can be useful to retrieve diagnostic error messages. Enabling it is optional. The command TMB:::setupRStudio() enables this feature and requires manual input to confirm.

## Linear regression example

We start by illustrating how TMB can be used to fit a simple linear model and how to handle parameters subject to constraints, which is particularly relevant for HMMs. A more comprehensive tutorial on how to use TMB can be found at https://kaskr.github.io/adcomp/ book/Tutorial.html.

Let ***x*** and ***y*** be datasets of size n, then the negative log-likelihood corresponding to a simple linear model is given by

*n*

−*l*(*a, b, σ*2) = − ) log(*φ*(*yi*; *a* + *bxi, σ*2)))*,* (1)

*i*=1

where *φ*(·; *µ, σ*2) is the density function of the univariate normal distribution with mean *µ* and variance *σ*2.

TMB requires the (negative) log-likelihood function to be written in C++ under a specific template, and then loaded into R. Minimization and other post-processing procedures are all carried out in R.

Therefore, we require 2 files. The first file named *linreg.cpp* is written in C++, and defines the negative log-likelihood (nll) function of the linear model as follows.

Biometrical Journal **52** (2020) 61 3

**# i n c l u d e** *<*TMB. hpp*> / / import t h e TMB t e m p l a t e*

**template** *<***c l a s s** Type*>*

Type o b j e c t i v e f u n c t i o n *<*Type *>* : : **operator** ( ) ( )

*{*

DATA VECTOR( y ) ; */ / Data v e c t o r y t r a n s m i t t e d from R*

DATA VECTOR( x ) ; */ / Data v e c t o r x t r a n s m i t t e d from R*

PARAMETER( a ) ; */ / Parameter a t r a n s m i t t e d from R* PARAMETER( b ) ; */ / Parameter b t r a n s m i t t e d from R* PARAMETER( log Sigma ) ; */ / Sigma on log−s c a l e t r a n s m i t t e d from R*

*/ / Transform log Sigma back t o n a t u r a l s c a l e*

Type sigma = exp ( log Sigma ) ;

*/ / Declare n e g a t i v e log−l i k e l i h o o d*

Type n l l = *−* sum ( dnorm ( y ,

a + b \* x , sigma ,

**t ru e** ) ) ;

*/ / This l e t s t h e user r e t r i e v e any v a r i a b l e s along with t h e i r s t andard e r r o r s*

ADREPORT( a ) ; ADREPORT( b ) ;

ADREPORT( sigma ) ;

**return** n l l ;

*}*

Note that we define data inputs *x* and *y* using the DATA VECTOR() declaration in the above code. Fur-

thermore, we declare the nll as a function of the three parameters a, b and log(*σ*). using the PARAMETER() declaration. Since the parameter *σ* has to be non-negative, the nll is parameterized in terms of log(*σ*) which can be freely estimated. This approach avoids the need of using constraint optimization methods. By sub- sequently calling the function ADREPORT on the variables, TMB can apply the generalized delta method described by [Kass and Steffey](#_bookmark30) ([1989](#_bookmark30)) to calculate the corresponding standard errors. This feature is partic- ularly useful when the likelihood has been reparameterized as above, and is especially relevant for HMMs. Following [Zucchini et al.](#_bookmark40) ([2016](#_bookmark40)), we will refer to the original parameters as natural parameters, and to their transformed version as the working parameters.

The second file needed is written in R and serves for compiling the nll function defined above, carrying out the estimation procedure by numerical optimization of the nll function and various post-processing of the results.

Once the C++ file with the negative log-likelihood is established it can be compiled and the objective function can be minimized as shown in the .R file below:

*# Setting up TMB*

**library**(TMB) TMB::**compile**("code/linreg.cpp")

## [1] 0

4 First Author*et al.* and dd: Running title

**dyn.load**(**dynlib**("code/linreg"))

*# Generate the data*

**set.seed**(123)

data <- **list**(y = **rnorm**(20) + 1:20, x = 1:20) parameters <- **list**(a = 0, b = 0, logSigma = 0)

*# Tell TMB to create the likelihood function*

obj\_linreg <- **MakeADFun**(data, parameters, DLL = "linreg",

silent = TRUE)

*# Optimization of the objective function with TMB* fit\_linreg <- **nlminb**(obj\_linreg$par, obj\_linreg$fn) fit\_linreg$par

##

a

b

logSigma

## 0.31009240 0.98395535 -0.05814659

The function sdreport will by default give the estimates and standard errors of the variables that the negative log-likelihood is parameterized in terms of:

**sdreport**(obj\_linreg)

##

##

##

##

##

##

sdreport(.) result

Estimate

a b

logSigma

0.31009240

0.98395535

-0.05814659

Std. Error 0.43829083

0.03658781

0.15811381

Maximum gradient component: 6.931683e-05

To get the estimates and standard errors of the variables called with ADREPORT() in the C++ file, we can run a summary call with the select = "report" option:

**summary**(**sdreport**(obj\_linreg), select = "report")

|  |  |  |  |
| --- | --- | --- | --- |
| ## |  | Estimate | Std. Error |
| ## | a | 0.3100924 | 0.43829083 |
| ## | b | 0.9839553 | 0.03658781 |
| ## | sigma | 0.9435116 | 0.14918222 |

And we can see the same estimates from a regular linear regression.

**lm**(y ˜ x, data = data)$coefficients

## (Intercept) x

## 0.3100925 0.9839554

# Estimation TMB/HMM

## Notation

A HMM is a model where the data is assumed to follow a mixture of distributions, going from one to the other as time progresses according to probabilities set by an underlying homogeneous

Biometrical Journal **52** (2020) 61 5

stationary Markov chain. In this paper we will focus on a Poisson-HMM, but only small changes to the code are needed to obtain models with other conditional distributions than the Poisson. The data denoted as {*Xn* : *n* = 1*, . . . , N* } is assumed to follow a mixture of *m* Poisson distributions with parameters {*λi* :

*i* = 1*, . . . , m*}.

Let {*Ct* : *t* = 1*, . . . , N* } be the underlying Markov chain. The Markov chain is assumed irreducible and aperiodic. [Grimmett et al.](#_bookmark28) ([2001](#_bookmark28), Lemma 6.3.5 on p. 225 and Theorem 6.4.3 on p. 227) has shown

that irreducibility ensures the existence of the stationary distribution. As has been shown by [Feller](#_bookmark24) ([1968](#_bookmark24),

p. 394), aperiodicity implies that a unique limiting distribution exists and is the stationary distribution.

*p*1(*x*) 0 

Let *pi*(*x*) = *P* (*Xt* = *x*|*Ct* = *i*)*,* ∀*i* = 1*, . . . , m* and let **P**(*x*) = 







*p*2(*x*)



. . .



.



*p*1(*x*1) *p*2(*x*1) *. . . pm*(*x*1)

We will call *p*1(*x*2) *p*2(*x*2) *. . . pm*(*x*2)

0 *pm*(*x*)

 ..





.



...

. . . ...

 the emission probability matrix. If *x* is a missing data,



*p*1(*xn*) *p*2(*xn*) *. . . pm*(*xn*)

we will define *pi*(*x*) = 1 and therefore the emission matrix of the dataset will contain a line filled with ones.

Furthermore, we let **Γ** = {*γij* } denote the transition matrix of the Markov chain, and ***δ*** be its stationary distribution.

Finally, we let *X*(*t*) = {*X*1*, . . . , Xt*} and *x*(*t*) = {*x*1*, . . . , xt*} denote the history up to time *t*.

In this paper we focus on Poisson HMMs, and so *pi*(*x*) = *e*

This can however be easily changed to other distributions.

*−λi x i*

*λ*

.

*x*!

## HMM Likelihood

It can be shown [(Zucchini et al.](#_bookmark40), [2016](#_bookmark40), p. 36) that the likelihood of an HMM is given by

*LN* = **P**(*X*(*t*) = *x*(*t*)) = ***δ*P**(*x*1)**ΓP**(*x*2)**ΓP**(*x*3) *. . .* **ΓP**(*xN* )**1**!

It is usual for HMMs to estimate the parameters ***θ*** (where ***θ*** = (*γ*11*, . . . , γ*1*m, . . . , γm*1*, . . . , γmm, λ*1*,*

*. . . , λm*)) subject to estimation in the case of a Poisson HMM) by optimizing a reparametrized version

of the log-likelihood log *L*(***ψ***), where ***ψ*** = *g*−1(***θ***) represent a set of unconstrained parameters. That transformation will be detailed below. As a result of the invariance principle, the maximum likelihood

estimate of ***θ*** is given by ***θ***ˆ = *g*(***ψ***ˆ ) where ***ψ***ˆ is the minimizer of − log *L*(***ψ***). To avoid constraints on the parameters, we transform the natural parameters {**Γ***,* ***λ***} into working parameters {**T***,* ***η***}.

In practice, as explained in [Section 2](#_bookmark0), we first create a set of natural parameters and turn them into

working parameters before feeding those in an optimizer. One possible transformation of **Γ** is done this way:

exp(*τij* )

*γij* = 1 +

*k*j=*i τik*

*,* for *i* /= *j*

Each row must add to 1, so we can get the diagonal elements of **Γ** easily.

*ηi* = log(*λi*).

The reverse transformation is

*τij* = log

/ *γij* \

1 − *k*j=*i γik*

= log(*γij/γii*)*,*

for *i* /= *j*

6 First Author*et al.* and dd: Running title

And *λi* = exp(*ηi*).

The relevant R function to do so is given by [Zucchini et al.](#_bookmark40) ([2016](#_bookmark40), p. 52) for the Poisson case.

*# Transform Poisson natural parameters to working parameters*

pois.HMM.pn2pw <- **function**(m, lambda, gamma, delta = **NULL**,

stationary = TRUE) {

tlambda <- **log**(lambda)

foo <- **log**(gamma / **diag**(gamma)) tgamma <- **as.vector**(foo[!**diag**(m)])

**if** (stationary) {

*# If tdelta is set to NULL and returned in the list,*

*# it will cause issues when optimizing with TMB*

**return**(**list**(tlambda = tlambda, tgamma = tgamma))

} **else** {

tdelta <- **log**(delta[- 1] / delta[1])

**return**(**list**(tlambda = tlambda, tgamma = tgamma, tdelta = tdelta))

}

}

Such a transformation can be adapted to other conditional distributions. Note that the parts concerning the Markov chain remain unchanged. The working parameters returned then serve as initial values for starting the optimization procedure of the likelihood via TMB. As illustrated in the previous section, we focus on unconstrained optimization, similar to CITE: Zucchini p.???. Hence, we parameterize the likelihood function as a function of the working parameters. The computation of the likelihood then requires to transform the working parameters back to natural parameters before calculating the negative log-likelihood via Equation (REFERENCE: likelihood function). This part is entirely set up in the C++ likelihood function.

Since the transformation of the TPM is identical for all HMMs, is is stored as a seperate C++ function Gamma w2n, , which is available in the file *utils.cpp* (See Appendix A.2 [item (iii)](#_bookmark19)).

This file also contains the function Delta w2n, which If necessary, we can define the C++ function turning the working stationary distribution vector into a natural parameter with available in the file utils.cpp. See Appendix A.2 [item (ii)](#_bookmark18). In practice, we didn’t use it.

Given a working parameter vector log lambda, the C++ function turning the working vector of the Poisson means into a natural parameter is much simpler:

v e c t o r *<*Type*>* lambda = t l ambda . exp ( ) ;

The “forward algorithm” allows for a recursive computation of the likelihood.

To state the forward algorithm let us define the vector ***α****t* for *t* = 1*,* 2*, . . . , N* so that

***α****t* = ***δ*P**(*x*1)**ΓP**(*x*2)**ΓP**(*x*3) *. . .* **ΓP**(*xt*)

*t*

= ***δ*P**(*x*1) n **ΓP**(*xs*)

*s*=2

= (*αt*(1)*, . . . , αt*(*m*))

where ***δ*** denotes the initial distribution of the Markov chain. By convention, the empty product is the identity matrix. It is called the forward algorithm because of the way the ***α****t* values are calculated

***α***0 = ***δ***

***α****t* = ***α****t*−1**ΓP**(*xt*) for *t* = 1*,* 2*, . . . , N LT* = ***α****T* **1**!

Biometrical Journal **52** (2020) 61 7

In the same way, the “backward algorithm” allows for a recursive computation of the likelihood.

To state the backward algorithm let us define the vector ***β****t* for *t* = 1*,* 2*, . . . , N* so that

***β***! !

*t* = **ΓP**(*xt*+1)**ΓP**(*xt*+2) *. . .* **ΓP**(*xT* ) *. . .* **1**

/ *N*

= n

*s*=*t*+1

\

**ΓP**(*xs*) **1**!

= (*βt*(1)*, . . . , βt*(*m*))

It is called the backward algorithm because of the way the ***β****t* values are calculated

***β****T* = **1**!

***β****t* = **ΓP**(*xt*+1)***β****t*+1 for *t* = *N* − 1*, N* − 2*, . . . ,* 1

*LT* = ***δβ***1

Computing those probabilities directly can lead to underflow errors, it is therefore better to maximize the log-likelihood instead. We chose to minimize the negative log-likelihood for convenience purposes. In addition, in our case **Γ** and ***λ*** are subject to some constraints:

1. The means *λi* of the state-dependent distributions must be non-negative, for *i* = 1*, . . . , m*
2. The parameters *γi,j* of the transition probability matrix **Γ** must be non-negative, and the rows of **Γ**

must add to one

It should be noted that we use a scaled version of the forward algorithm in order to calculate the likeli- hood, as suggested by [Zucchini et al.](#_bookmark40) ([2016](#_bookmark40), p. 48).

In summary, we need to define the state-dependent probabilities (from Poisson distributions in our case) and the transformations in R and C++ (they depend on the distributions used, but can be easily adapted). We show below the next step: writing the likelihood function in C++ and using it to model a dataset.

## Using TMB

* + 1. **Likelihood function**

As we did with the linear regression example, we first need to define our objective function: the negative log-likelihood, in a C++ file. Let’s name it poi hmm.cpp. Its code can be found in the Appendix A.2 [item (i)](#_bookmark17).

## Optimization

Once the objective function is written in C++, we can tell R to optimize its parameters.

1. Loading packages

*# Load TMB and optimization packages*

**library**(TMB) **library**(optimr)

*# Run the C++ file containing the TMB code*

TMB::**compile**("code/poi\_hmm.cpp", "-O1 -g", DLLFLAGS = "")

8 First Author*et al.* and dd: Running title

## [1] 0

*# Load it*

**dyn.load**(**dynlib**("code/poi\_hmm"))

*# Load the parameter transformation function*

**source**("functions/utils\_no\_tmb.R")

1. Loading the dataset, see [Section 7.1](#_bookmark10) for more details.

**load**("data/fetal-lamb.RData") lamb\_data <- lamb

1. Creating natural parameters as initial values for the optimizer.

We use the best amount of hidden states (2) according to the BIC that [Leroux and Puterman](#_bookmark32) ([1992](#_bookmark32)) reported. Although the AIC selects 3 hidden states, our goal is to show a simple example.

*# Model with 2 states*

m <- 2

TMB\_data <- **list**(x = lamb\_data, m = m)

*# 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)

1. Transforming them to working parameters

*# Turn them into working parameters*

parameters <- **pois.HMM.pn2pw**(m, lambda, gamma)

1. Creating the TMB likelihood function object

obj\_tmb <- **MakeADFun**(TMB\_data, parameters, DLL = "poi\_hmm",

silent = TRUE)

1. Optimizing

mod\_tmb <- **nlminb**(start = obj\_tmb$par, objective = obj\_tmb$fn)

*# Check that it converged successfully*

mod\_tmb$convergence == 0

## [1] TRUE

We are using nlminb because it is the fastest we have tested. Details are available in [item (iii)](#_bookmark13).

1. Getting the ML estimates of the natural parameters and their standard errors, sent by ADREPORT

It should be noted that the gamma parameter is shown column-wise below.

Biometrical Journal **52** (2020) 61 9

|  |  |  |  |
| --- | --- | --- | --- |
| ## |  | Estimate | Std. Error |
| ## | lambda | 0.25636530 | 0.04016451 |
| ## | lambda | 3.11475069 | 1.02131434 |
| ## | gamma | 0.98872130 | 0.01063573 |
| ## | gamma | 0.31033874 | 0.18468645 |
| ## | gamma | 0.01127870 | 0.01063573 |
| ## | gamma | 0.68966126 | 0.18468645 |
| ## | delta | 0.96493132 | 0.03181447 |
| ## | delta | 0.03506868 | 0.03181447 |

1. It is possible to tell the optimizer to use the gradient and/or hessian computed by TMB:

**summary**(**sdreport**(obj\_tmb), "report")

*# The negative log-likelihood is accessed by the objective*

*# attribute of the optimized object*

mod\_tmb$objective

## [1] 177.5188

mod\_tmb <- **nlminb**(start = obj\_tmb$par, objective = obj\_tmb$fn, gradient = obj\_tmb$gr, hessian = obj\_tmb$he)

mod\_tmb$objective

## [1] 177.5188

The use of an exact gradient and hessian provided by TMB can make the model fit better, although in our case it doesn’t make a difference.

The dataset used is provided by [Leroux and Puterman](#_bookmark32) ([1992](#_bookmark32)). Although our estimates are close to the paper’s estimates, they’re not exactly the same. The reason is that they use a modified likelihood and we don’t. This is clear when comparing the likelihoods with only 1 state.

A 1 state Poisson HMM is the same as a Poisson regression model, for which the log-likelihood has the expression

*l*(*λ*) = log

/ *n xi*

n

*λ*

*e*−*λ* \

*i*=1

*xi*!

/ *n* \ *n*

)

= −*nλ* + log(*λ*)

) *xi*

*i*=1

— log(*xi*!)*.*

*i*=1

They find a ML estimate *λ* = 0*.*3583 and a log-likelihood of -174.26.

x <- lamb\_data n <- **length**(x) l <- 0.3583

- n \* l + **log**(l) \* **sum**(x) - **sum**(**log**(**factorial**(x)))

## [1] -201.0436

10 First Author*et al.* and dd: Running title

We can see that the log-likelihood is different, but when we remove the constant − *n*

*i*=1

log(*xi*!), it

becomes

- n \* l + **log**(l) \* **sum**(x)

## [1] -174.2611

In our estimation procedures, we use the complete formula for the log-likelihood. Therefore our likeli- hood differs sligthly from the paper.

## Computing the stationary distribution

In the objective function above in [Section 3.3.1](#_bookmark3), we calculated the stationary distribution of the *m* state HMM’s Markov chain with transition probability matrix **Γ**, but we skipped over the code to do that.

[Zucchini et al.](#_bookmark40) ([2016](#_bookmark40)) shows that to calculate the stationary distribution, we can solve this equation for

***δ***:

***δ***(**I***m* − **Γ** + **U**) = **1**

where **I***m* is the *m* ∗ *m* identity matrix, **U** is a *m* ∗ *m* matrix of ones, and **1** is a row vector of ones. An implementation of this in R is

*# Compute the stationary distribution of a Markov chain*

*# with transition probability gamma*

stat.dist <- **function**(gamma) {

m <- **dim**(gamma)[1]

**return**(**solve**(**t**(**diag**(m) - gamma + 1), **rep**(1, m)))

}

More details are available in .

In order to use it in TMB, we show a way to implement it in C++ in the file utils.cpp, see Appendix A.2 [item (iv)](#_bookmark20).

## Nested model specification

Nested models can be useful for multiple reasons. For example, there can be a need to fix some parameters because of some biological or physical constraints.

In our situation, using a nested model solves this issue at the cost of having a worse fit. The reason is that some parameters are fixed and the others’ estimates have different values than in the original model. Since the estimates can be different, the fit worsens.

To showcase the advantage of nested models, we will reuse the 2 state model.

TMB can be instructed to treat some parameters as constants. We will arbitrarily fix *λ*1 to 1.

*# Get the previous values, and fix some* fixed\_par\_lambda <- lambda fixed\_par\_lambda[1] <- 1 fixed\_par\_gamma <- gamma

Since we can only fix working parameters, we chose to fix a natural parameter for which we can eas- ily identify the corresponding working parameter to fix, for simplicity. Fixing a value in the transition probability matrix would be tricky, so we chose a Poisson mean instead.

Biometrical Journal **52** (2020) 61 11

Now that the nested model parameters have been made, we need to transform them into a set of working parameters.

*# Transform them into working parameters*

new\_parameters <- **pois.HMM.pn2pw**(m = m,

lambda = fixed\_par\_lambda, gamma = fixed\_par\_gamma)

In order to have TMB treat parameters as constants, the map argument of the MakeADFun function must be a list. This list must contain named vectors filled with NA if the parameter is to be treated as a constant, or factor levels.

Equal factor level is collected to a common value, so we want 1 different factor level for each non-fixed parameter, and NA for each fixed parameter. We take increasing numbers to make factors, and replace with NA where necessary.

map <- **list**(tlambda = **as.factor**(**c**(NA, 2)), tgamma = **as.factor**(**c**(1, 3)))

*# The map is fed to the MakeADFun function*

fixed\_par\_obj\_tmb <- **MakeADFun**(TMB\_data, new\_parameters,

DLL = "poi\_hmm", silent = TRUE, map = map)

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)

Old estimates and standard errors, and the new ones

Before After

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Estimate | Std. Error | Estimate | Std. Error |
| *λ*1 | 0.26 | 0.04 | 1.00 | 0.00 |
| *λ*2 | 3.11 | 1.02 | 3.44 | 0.86 |
| *γ*1*,*1 | 0.99 | 0.01 | 1.00 | 0.00 |
| *γ*2*,*1 | 0.31 | 0.18 | 0.19 | 0.18 |
| *γ*1*,*2 | 0.01 | 0.01 | 0.00 | 0.00 |
| *γ*2*,*2 | 0.69 | 0.18 | 0.81 | 0.18 |
| *δ*1 | 0.96 | 0.03 | 0.98 | 0.03 |

*δ*2 0.04 0.03 0.02 0.03

**Table 1** Before and after fixing low parameters to 0 and the likelihood got worse as expected.

*# The negative log-likelihood is accessed by the attribute*

*# "objective" of the optimized object*

mod\_tmb$objective

## [1] 177.5188

12 First Author*et al.* and dd: Running title

fixed\_par\_mod\_tmb$objective

## [1] 264.1636

It should be noted that some inconsistencies can happen. For example, the stationary distribution is a vector of probabilities and should sum to 1. However, we can see that it doesn’t exactly sum to 1.

adrep <- **summary**(**sdreport**(obj\_tmb), "report")

estimate\_delta <- adrep[**rownames**(adrep) == "delta", "Estimate"]

**sum**(estimate\_delta)

## [1] 1

**sum**(estimate\_delta) == 1

## [1] FALSE

This is likely due to machine approximations when making numbers far apart from each other interact together. For example, in R a small number is not 0 but is treated as 0 when added to a much larger number.

1e-100 == 0

## [1] FALSE

(1 + 1e-100) == 1

## [1] TRUE

This can result in inconsistencies when numbers far apart from each other are involved. Fortunately, this doesn’t seem to cause any issues.

# Confidence intervals

## Hessian based confidence intervals

First, let us consider evaluation of parameter uncertainty via the Hessian. As the Hessian ∇2 log *L*(*ψ*ˆ) refers to the transformed parameters *ψ* the delta-method must be used to obtain an estimate of the covari- ance matrix of *θ*ˆ:

Σ*θ*ˆ = −∇*g*(*ψ*ˆ) ∇

( 2

log *L*(*ψ*ˆ)

'−1

∇*g*(*ψ*ˆ)! (2)

With minimal effort from the users perspective, TMB can be instructed to calculate Σ*θ*ˆ (by automatic differentiation). Standard errors of derived parameters, such as the stationary distribution ***δ*** which is a function of **Γ**, can be calculated by the delta-method similarly.

TMB provides an easy way to retrieve these. Following the example above, as we saw in [Section 3.3.2](#_bookmark4),

*# Get all standard errors*

adrep <- **summary**(**sdreport**(obj\_tmb), "report") adrep

Biometrical Journal **52** (2020) 61 13

|  |  |  |  |
| --- | --- | --- | --- |
| ## |  | Estimate | Std. Error |
| ## | lambda | 0.25636541 | 0.04016445 |
| ## | lambda | 3.11475432 | 1.02131181 |
| ## | gamma | 0.98872128 | 0.01063571 |
| ## | gamma | 0.31033853 | 0.18468648 |
| ## | gamma | 0.01127872 | 0.01063571 |
| ## | gamma | 0.68966147 | 0.18468648 |
| ## | delta | 0.96493123 | 0.03181445 |
| ## | delta | 0.03506877 | 0.03181445 |

It should be noted that although the estimates can be found in the optimization variable opt in their working form, they are also sent to the MakeADFun object obj and can be retrieved as shown above. sdreport shows information about the working parameters’ estimates, whereas the summary of that report will also show information about the variables reported by the function ADREPORT.

*# More help with ?summary.sdreport*

We will now see how to get Wald confidence intervals as definded by [Wald](#_bookmark39) ([1943](#_bookmark39)) using TMB. With our 2 state Poisson HMM, the 100(1 − *α*)% confidence interval for *a* is *a* ± *z*1−*α/*2 ∗ *σa* where *zx* is the *x*-percentile of the standard normal distribution, and *σa* is the standard error of *a*.

|  |  |  |  |
| --- | --- | --- | --- |
| ## |  | lower\_bound | upper\_bound |
| ## | lambda | 0.177644535 | 0.33508628 |
| ## | lambda | 1.113019955 | 5.11648869 |
| ## | gamma | 0.967875672 | 1.00956689 |
| ## | gamma | -0.051640322 | 0.67231737 |
| ## | gamma | -0.009566886 | 0.03212433 |
| ## | gamma | 0.327682625 | 1.05164032 |
| ## | delta | 0.902576056 | 1.02728641 |
| ## | delta | -0.027286406 | 0.09742394 |

Estimates of ***θ*** and ***δ***, with accompanying confidence intervals are displayed in [Table 12](#_bookmark15) and [Table 13](#_bookmark16).

*# sdreport gives the estimates of the parameters*

*# and their standard errors*

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 upper\_bound <- estimates + q95\_norm

*# Show the CI*

**cbind**(lower\_bound, upper\_bound)

\*

\*

std\_errors std\_errors

For comparison, we have included the corresponding standard deviations resulting from replacing ∇2 log *L*(*ψ*ˆ)

in [(Equation 2](#_bookmark6)) with traditional numerical approximations using the R functions nlm (nlmb) and fdHess

14 First Author*et al.* and dd: Running title

(nlme). The difference when using these approximations are of order 1*e* − 05 or less, however one would still have to calculate ∇*g*(*ψ*ˆ) for these approximations to be useful.

As mentioned earlier, for a larger amount of hidden states, TMB may be unable to give some or any standard standard errors because some variables are close to their boundaries.

In that situation, using a nested model might solve this issue at the cost of the model usually having a worse fit.

Refer to [Section 3.5](#_bookmark5) for information on how to make a nested model using TMB.

## Likelihood profile based confidence intervals

Next, we consider evaluating uncertainty using likelihood-profiles. Assuming normality of the maximum likelihood (ML) estimators, confidence intervals for the parameters can be obtained using the above esti- mates of standard deviations. For a fixed sample size, the validity of the normality assumption can often be violated and in these cases a Likelihood-based CI is more robust (reference). Let *η* be single parameter in a model with parameters (*η, θ*) and likelihood *L*(*η, θ*), and let *Lp*(*η*) be the profile likelihood defined by *Lp*(*η*) = max*θ L*(*η, θ*). Then a likelihood-based CI for *η* is given by

(

*η* : 2 log(

*L*(*η*ˆ*, θ*ˆ) *Lp*(*η*)

2

*< χ*

1*,*(1−*α*)

(3)

where *χ*2

1*,*(1−*α*)

is the 1 − *α* quantile of a *χ*2 distribution with 1 degree of freedom. TMB allows for

very efficient computation of both the profile likelihood *L*(*η*) and the CI given by [(Equation 3](#_bookmark7)), and

this has been used to produce [Table 12](#_bookmark15) and [Table 13](#_bookmark16) which display the profile log-likelihood and the corresponding likelihood-based CI’s in the model for the lamb and the simulated dataset. Note that the problem of transformed parameters is easier to deal with when making likelihood-based CI’s, since

{*g*(*η*)*, Lp*(*g*(*η*))} = {*g*(*η*)*, Lp*(*η*)} for any one-to-one function *g* (invariance principle).

Again, TMB provides an easy way to obtain those intervals. The name argument allows to profile pa- rameter. The trace argument indicates how much information on the optimization you want the function to show. Following the HMM example, if we wish to profile the 2*nd* working parameter log lambda2, we have to feed its position into the name argument:

profile <- **tmbprofile**(obj = obj\_tmb,

name = 2, trace = FALSE)

*# Confidence interval of lambda*

**exp**(**confint**(profile))

##

lower

upper

## tlambda 1.265339 4.947733

It should be noted that the argument lincomb allows to profile any linear combination of parameters by using a vector of the linear coefficients in the same order as the parameters.

This method can sometimes give NA values, usually when a ML estimate is close to a boundary, but not only. For example, the first working parameter log lambda is pretty low (−4*.*47). It becomes too difficult to profile the likelihood so the function fails to provide a meaningful confidence interval.

Another important issue is about profiling with a univariate model. With only 1 (hidden) state, a Poisson HMM becomes a univariate Poisson regression model. Therefore, the profile becomes a plot of the likeli- hood when the Poisson mean varies. However, tmbprofile fails to provide a confidence interval. The

Biometrical Journal **52** (2020) 61 15

reason is likely that it tries to optimize the likelihood despite the lack of parameters to change, and there- fore fails. [Visser et al.](#_bookmark38) ([2000](#_bookmark38)), [Meeker and Escobar](#_bookmark35) ([1995](#_bookmark35)) and [Venzon and Moolgavkar](#_bookmark37) ([1988](#_bookmark37)) provide more details on profiling likelihoods.

## Bootstrap based confidence intervals

Finally, we consider evaluating uncertainty using bootstrap. There are many ways to bootstrap. Non- parametric bootstraping of time series is possible, as [Ha¨rdle et al.](#_bookmark29) ([2003](#_bookmark29)) has shown. But since our focus is not on bootstrapping techniques, we chose a parametric approach. See [Efron and Tibshirani](#_bookmark23) ([1993](#_bookmark23)) for more details on bootstrapping.

From the parameters’ ML estimates, we generate new data and re-estimate the parameters 500 times. From that list of new estimates we can get the 2.5th and 97.5th percentiles and get 95% confidence intervals for the parameters.

We show below how we get confidence intervals using bootstrap, based on the 3 state Poisson HMM estimates from above.

1. First, we need a function to generate random data from a HMM.

*# Generate a random sample from a HMM*

pois.HMM.generate\_sample <- **function**(ns, mod) {

mvect <- 1:mod$m state <- **numeric**(ns)

state[1] <- **sample**(mvect, 1, prob = mod$delta)

**for** (i **in** 2:ns) {

state[i] <- **sample**(mvect, 1, prob = mod$gamma[state[i - 1], ])

}

x <- **rpois**(ns, lambda = mod$lambda[state])

**return**(x)

}

1. Then, when the model is estimated each time, we don’t impose an order for the states. This can lead to the label switching problem, where states aren’t ordered the same way in each model. To address this, we re-ordered the states by ascending Poisson means.

Sorting the means is pretty straightforward. Re-ordering the TPM is a little trickier. To do so, we took the permutations of the states given by the sorted Poisson means, and permuted each row index and column index to its new value.

The function we used is

*# Relabel states by increasing Poisson means*

pois.HMM.label.order <- **function**(m, lambda, gamma, delta = **NULL**) {

*# Get the indexes of the sorted states*

*# according to ascending lambda*

sorted\_lambda <- **sort**(lambda, index.return = TRUE)$ix

*# Re-order the TPM according to the switched states*

*# in the sorted lambda*

ordered\_gamma <- **matrix**(0, nrow = m, ncol = m)

**for** (col **in** 1:m) {

new\_col <- **which**(sorted\_lambda == col)

**for** (row **in** 1:m) {

16 First Author*et al.* and dd: Running title

new\_row <- **which**(sorted\_lambda == row) ordered\_gamma[row, col] <- gamma[new\_row, new\_col]

}

}

*# Re-order the stationary distribution if it was provided*

*# Generate it otherwise*

**if** (**is.null**(delta)) {

delta <- **stat.dist**(ordered\_gamma)

} **else** {

delta <- delta[sorted\_lambda]

}

**return**(**list**(lambda = **sort**(lambda),

gamma = ordered\_gamma, delta = delta))

}

Let’s show an example to understand the process. For readability, the TPM is filled with row and column indexes instead of probabilities.

lambda <- **c**(30, 10, 20)

gamma <- **matrix**(**c**(11, 12, 13,

21, 22, 23,

31, 32, 33), byrow = TRUE, ncol = 3)

**pois.HMM.label.order**(m = 3, lambda, gamma)

##

##

##

##

##

##

##

##

##

##

##

$lambda [1] 10 20

30

$gamma

[,1]

[1,]

[2,]

[3,]

33

13

23

[,2]

31

11

21

[,3]

32

12

22

$delta

[1] -0.032786885

0.016393443 -0.008196721

State 1 has been relabeled state 3, state 2 became state 1, and state 3 became state 2.

In short, 1 → 3*,* 2 → 1*,*and3 → 2.

1. Bootstrap code

bootstrap\_estimates <- **data.frame**() DATA\_SIZE <- **length**(lamb\_data)

*# Set how many parametric bootstrap samples we create*

BOOTSTRAP\_SAMPLES <- 10

*# ML parameters*

ML\_working\_estimates <- obj\_tmb$env$last.par.best

Biometrical Journal **52** (2020) 61 17

ML\_natural\_estimates <- obj\_tmb$**report**(ML\_working\_estimates) gamma <- ML\_natural\_estimates$gamma

lambda <- ML\_natural\_estimates$lambda delta <- ML\_natural\_estimates$delta

*# Parameters for TMB*

cols <- **names**(ML\_working\_estimates)

tgamma <- ML\_working\_estimates[cols == "tgamma"] tlambda <- ML\_working\_estimates[cols == "tlambda"] ML\_TMB\_parameters <- **list**(tlambda = tlambda,

tgamma = tgamma) params\_names <- **c**("lambda", "gamma", "delta")

**for** (idx\_sample **in** 1:BOOTSTRAP\_SAMPLES) {

*# Loop as long as there is an issue with nlminb*

## repeat {

*#simulate the data*

bootstrap\_data <- **pois.HMM.generate\_sample**(DATA\_SIZE,

**list**(m = m,

lambda = lambda, gamma = gamma, delta = delta))

*# Parameters for TMB*

TMB\_data\_bootstrap <- **list**(x = bootstrap\_data, m = m)

*# Estimate the parameters*

obj <- **MakeADFun**(TMB\_data\_bootstrap,

ML\_TMB\_parameters, DLL = "poi\_hmm", silent = TRUE)

*# Using tryCatch to break loop if an error happens*

*# will interrupt the outer loop, not the inner one*

mod\_temp <- **NULL**

**try**(mod\_temp <- **nlminb**(start = obj$par, objective = obj$fn,

gradient = obj$gr, hessian = obj$he),

silent = TRUE)

*# If nlminb doesn't reach any result, retry*

**if** (**is.null**(mod\_temp)) {

## next

}

*# If nlminb doesn't converge successfully, retry*

**if** (mod\_temp$convergence != 0) {

## next

}

working\_parameters <- obj$env$last.par.best

18 First Author*et al.* and dd: Running title

natural\_parameters <- obj$**report**(working\_parameters)

l <- natural\_parameters$lambda g <- natural\_parameters$gamma d <- natural\_parameters$delta

*# Label switching*

natural\_parameters <- **pois.HMM.label.order**(m, l, g, d)

*# If some parameters are NA for some reason, retry*

**if** (**anyNA**(natural\_parameters[params\_names], recursive = TRUE)) {

## next

}

*# If everything went well, end the "repeat" loop*

## break

}

*# The values from gamma are taken columnwise* natural\_parameters <- **unlist**(natural\_parameters[params\_names]) len\_par <- **length**(natural\_parameters)

bootstrap\_estimates[idx\_sample, 1:len\_par] <- natural\_parameters

}

*# Lower and upper 95% bounds*

q <- **apply**(bootstrap\_estimates, 2, **function**(par\_estimate) {

**quantile**(par\_estimate, probs = **c**(0.025, 0.975))

})

params\_names <- **paste0**(**rep**("lambda", m), 1:m)

*# Get row and column indexes for gamma instead of the default*

*# columnwise index: the default indexes are 1:m for the 1st column,*

*# then (m + 1):(2 \* m) for the 2nd, etc...*

**for** (gamma\_idx **in** 1:m ˆ 2) {

row <- (gamma\_idx - 1) %% m + 1 col <- (gamma\_idx - 1) %/% m + 1 row\_col\_idx <- **c**(row, col) params\_names <- **c**(params\_names,

**paste0**("gamma",

**paste0**(row\_col\_idx, collapse = "")))

}

params\_names <- **c**(params\_names,

**paste0**(**rep**("delta", m), 1:m)) bootstrap\_CI <- **data.frame**("Parameter" = params\_names,

"Estimate" = **c**(lambda, gamma, delta), "Lower bound" = q[1, ],

"Upper bound" = q[2, ])

**print**(bootstrap\_CI, row.names = FALSE)

## Parameter Estimate Lower.bound Upper.bound

## lambda1 0.25636541 7.117339e-09 0.2954041

Biometrical Journal **52** (2020) 61 19

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ## | lambda2 | 3.11475432 | 3.565353e-01 | 3.4379317 |
| ## | gamma11 | 0.98872128 | 4.444232e-01 | 0.9947810 |
| ## | gamma21 | 0.31033853 | 9.937752e-02 | 0.5693026 |
| ## | gamma12 | 0.01127872 | 5.219006e-03 | 0.5555768 |
| ## | gamma22 | 0.68966147 | 4.306974e-01 | 0.9006225 |
| ## | delta1 | 0.96493123 | 2.635814e-01 | 0.9830019 |
| ## | delta2 | 0.03506877 | 1.699805e-02 | 0.7364186 |

1. It should be noted that some estimates can be very large or small. This can happen when the randomly generated bootstrap sample contains long chains of the same values. However, a large number of bootstrap samples lowers that risk since we ”leave out” 5% of the extreme values in the CI. It could be useful to skip the bootstrap samples where the estimates are too far out of expectations.

# State inference

## Setup

Given an optimized MakeADFun object obj, we need to setup some variables to compute the probabilities detailed below.

*# Retrieve the objects at ML value*

adrep <- obj\_tmb$**report**(obj\_tmb$env$last.par.best) delta <- adrep$delta

gamma <- adrep$gamma

emission\_probs <- adrep$emission\_probs n <- adrep$n

m <- **length**(delta) mllk <- adrep$mllk

## Log-forward probabilities

The forward probabilities have been detailed in [Section 3.2](#_bookmark2). We show here a way to compute the log of the forward probabilities, using a scaling scheme defined by [Zucchini et al.](#_bookmark40) ([2016](#_bookmark40)).

*# Compute log-forward probabilities (scaling used)*

lalpha <- **matrix**(NA, m, n)

foo <- delta \* emission\_probs[1, ] sumfoo <- **sum**(foo)

lscale <- **log**(sumfoo) foo <- foo / sumfoo

lalpha[, 1] <- **log**(foo) + lscale

**for** (i **in** 2:n) {

foo <- foo %\*% gamma \* emission\_probs[i, ] sumfoo <- **sum**(foo)

lscale <- lscale + **log**(sumfoo) foo <- foo / sumfoo

lalpha[, i] <- **log**(foo) + lscale

}

*# lalpha contains n=240 columns, so we only display 5 for readability*

20 First Author*et al.* and dd: Running title

lalpha[, 1:5]

## [,1]

[,2]

[,3]

[,4]

[,5]

## [1,] -0.2920639 -0.5591179 -0.8265948 -1.094088 -1.361583

## [2,] -6.4651986 -7.7716769 -8.1146145 -8.385232 -8.652850

## Log-backward probabilities

The backward probabilities have been defined in the same section.

*# Compute log-backwards probabilities (scaling used)*

lbeta <- **matrix**(NA, m, n) lbeta[, n] <- **rep**(0, m) foo <- **rep** (1 / m, m) lscale <- **log**(m)

**for** (i **in** (n - 1):1) {

foo <- gamma %\*% (emission\_probs[i + 1, ] \* foo) lbeta[, i] <- **log**(foo) + lscale

sumfoo <- **sum**(foo) foo <- foo / sumfoo

lscale <- lscale + **log**(sumfoo)

}

*# lbeta contains n=240 columns, so we only display 4 for readability*

lbeta[, 1:4]

##

[,1]

[,2]

[,3]

[,4]

## [1,] -177.2275 -176.9600 -176.6925 -176.4250

## [2,] -178.3456 -178.0781 -177.8099 -177.5253

## Smoothing probabilities

The smoothing probabilities are defined in [Zucchini et al.](#_bookmark40) ([2016](#_bookmark40)) as *P* (*Ct* = *i*|*X*(*N* ) = *x*(*N* )) = *αt* (*i*)*βt* (*i*) .

*LT*

*# Compute conditional state probabilities, smoothing probabilities*

stateprobs <- **matrix**(NA, ncol = n, nrow = m) llk <- - mllk

**for**(i **in** 1:n) {

stateprobs[, i] <- **exp**(lalpha[, i] + lbeta[, i] - llk)

}

*# Most probable states*

ldecode <- **rep**(NA, n)

**for** (i **in** 1:n) {

ldecode[i] <- **which.max**(stateprobs[, i])

}

ldecode

## [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

## [32] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Biometrical Journal **52** (2020) 61 21

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ## | [63] | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 1 |
| ## | [94] | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ## | [125] | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ## | [156] | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ## | [187] | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ## | [218] | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |  |  |  |  |  |  |  |

## Forecast, h-step-ahead-probabilities

The forecast distribution or h-step-ahead-probabilities as well as its implementation in R is detailed in [Zucchini et al.](#_bookmark40) ([2016](#_bookmark40)).

Let ***ϕ****N* = ***α****N* .

***α****N* **1***!*

Then,

*P* (*XN* +*h* = *x*|*X*(*N* ) = *x*(*N* )) =

***α****N* **Γ***h***P**(*x*)**1**!

***α****N* **1**!

= ***ϕ****N* **Γ***h***P**(*x*)**1**!

An implementation of this, using a scaling scheme is

*# Number of steps*

h <- 1

*# Values for which we want the forecast probabilities*

xf <- 0:50

nxf <- **length**(xf)

dxf <- **matrix**(0, nrow = h, ncol = nxf) foo <- delta \* emission\_probs[1, ] sumfoo <- **sum**(foo)

lscale <- **log**(sumfoo) foo <- foo / sumfoo

**for** (i **in** 2:n) {

foo <- foo %\*% gamma \* emission\_probs[i, ] sumfoo <- **sum**( foo)

lscale <- lscale + **log**(sumfoo) foo <- foo / sumfoo

}

emission\_probs\_xf <- **get.emission.probs**(xf, lambda)

**for** (i **in** 1:h) {

foo <- foo %\*% gamma

**for** (j **in** 1:m) {

dxf[i, ] <- dxf[i, ] + foo[j] \* emission\_probs\_xf[, j]

}

}

*# dxf contains n=240 columns, so we only display 4 for readability*

dxf[, 1:4]

## [1] 0.765294660 0.197684763 0.027659770 0.004772415

22 First Author*et al.* and dd: Running title

## Global decoding using the Viterbi algorithm

The Viterbi algorithm is detailed in [Zucchini et al.](#_bookmark40) ([2016](#_bookmark40)). It calculates the sequence of states (*i*∗*, . . . , i*∗ )

1 *N*

which maximizes the conditional probability of all states simultaneously, i.e.

(*i*∗*, . . . , i*∗ ) = arg max

*P* (*C*1 = *i*1*, . . . , CN* = *iN* |*X*(*N* ) = *x*(*N* ))

1 *N*

*i*1 *,...,iN* ∈{1*,...,m*}

An implementation of it is

xi <- **matrix**(0, n, m)

foo <- delta \* emission\_probs[1, ] xi[1, ] <- foo / **sum**(foo)

**for** (i **in** 2:n) {

foo <- **apply**(xi[i - 1, ] \* gamma, 2, max) \* emission\_probs[i, ] xi[i, ] <- foo / **sum**(foo)

}

iv <- **numeric**(n)

iv[n] <- **which.max**(xi[n, ])

**for** (i **in** (n - 1):1){

iv[i] <- **which.max**(gamma[, iv[i + 1]] \* xi[i, ])

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| }  iv |  | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ## | [1] | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ## | [32] | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ## | [63] | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 1 |
| ## | [94] | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ## | [125] | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ## | [156] | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ## | [187] | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ## | [218] | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |  |  |  |  |  |  |  |

# Application to hospital data

## Hospital data description

We timed estimations using different parameters on a dataset provided by a hospital from Assistance publique – Hoˆpitaux de Paris, a french hospital trust.

The last dataset is the number of patients who entered a hospital each hour between the first hour of 2010 and the last hour of 2019. The data was anonymized and grouped using the package dplyr. That data contains 87648 data points.

The zeroes in the column PATIENTS indicate that no one arrived at the hospital during the hour. The column DATE is a datetime item useful for sorting and plotting the data.

The columns YEAR is an integer taking values 2010, 2011, . . . , 2019.

The column PATIENTS represents the number of people arriving at the hospital between the hour men- tioned and the next.

For example, the dataset starts on Januray 1*st* 2010 on the hour 0 with an amount of patients of 6, and

indicates that 6 people arrived between 00:00 and 00:59.

The column WDAY represents the number of the day in the week: 1 is Monday and 7 is Sunday.

Biometrical Journal **52** (2020) 61 23

## Speed comparison

Different numbers of hidden states *m* were used: (1*,* 2 for the lamb dataset, 1*,* 2*,* 3 for the simulated dataset, and 1*,* 2*,* 3*,* 4 for the hospital dataset) to compare speeds. Overall, the acceleration grows with the amount of hidden states *m* and the size of the dataset. The lamb dataset and the simulated one will be used in [Section 7](#_bookmark9). We will focus on the hospitals patient arrivals in this section.

For readability, DM denotes direct maximization without using TMB. TMB1, ..., TMB4 denotes di- rect maximization using TMB with and without the exact gradient and hessian. The following table will summarize the notation

**Table 2** Naming of TMB parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | TMB1 | TMB2 | TMB3 | TMB4 |
| Exact gradient is used  Exact hessian is used | No  No | No  Yes | Yes  No | Yes  Yes |

The following [Figure 1](#_bookmark8) below shows the time taken for estimating an HMM with and without making use of TMB on the hospital dataset through boxplots for *m* = 1*,* 2*,* 3*,* 4 hidden states, on a log-scale.

Parameter estimation time

**Hospital data, size = 87648**

100.0

10.0

Time (seconds)

1.0

0.1

1 2 3 4

DM TMB1 TMB2 TMB3 TMB4 DM TMB1 TMB2 TMB3 TMB4 DM TMB1 TMB2 TMB3 TMB4 DM TMB1 TMB2 TMB3 TMB4

Exact/inexact gradient and hessian

**Figure 1** Time w/o using TMB with hospital data

As could have been guessed, the time it takes to estimate a model’s parameters increases as the com- plexity of the model (*m*) increases. It seems using TMB accelerates the estimation, as can be seen by the drop in time from DM to any estimation using TMB. Interestingly, for *m >* 1, TMB3 shows a clear acceleration when compared to other estimations using TMB. The reason is unclear.

24 First Author*et al.* and dd: Running title

Each box of the boxplot doesn’t show a lot of variation, so looking at the percentage decrease of the mean times for TMB when compared to the mean times for DM will summarize this graph nicely. Instead of time decreases, we look at speed increases from using TMB on the hospital dataset and see that the highest acceleration when using TMB happens for TMB3 and is 4565%.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| m | TMB1 | TMB2 | TMB3 | TMB4 |
| 1 | 1444 | 1461 | 1205 | 1198 |
| 2 | 1543 | 1529 | 2653 | 2134 |
| 3 | 1112 | 1044 | 3519 | 1373 |
| 4 | 737 | 750 | 4565 | 1174 |

**Table 3** Speed percentage increase of estimation using TMB on the hospital dataset

## Interpretation

Looking at a boxplot of the data lets us discover the hourly distribution of the data. We chose to use hours instead of days or other time durations because the difference in arrivals between night and day was already obvious for the doctors themselves, and we would likely find at least two distinct regimes there.

**Hourly arrivals**

40



20

PATIENTS

0

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23

HOUR

We can see that more patients arrive in the day rather than during the night as was guessed, but also that more patients arrive in the afternoon rather than in the evening. Also, most patients show up in the early part of the day and around noon. Given the amount of data (87648/24 = 3652 data points for each hour), we can apply a z-test to see if the means are different.

Biometrical Journal **52** (2020) 61 25

We find a p-value approximated to 0, so the test shows that the difference between the average amount of patient arrival between 6 and 6:59am and the one between 10 and 10:59am is statistically significant. This agrees with the conclusion of the doctors and shows that there are likely at least 2 different distributions in the hourly arrivals of patients, justifying an educated guess of 2 or more hidden states.

To choose more accurately the number of hidden states, we will try a few, and look at the quality of each fit.

|  |  |  |  |
| --- | --- | --- | --- |
| m | nll | AIC | BIC |
| 1 | 324288 | 648578 | 648587 |
| 2 | 259295 | 518598 | 518636 |
| 3 | 247403 | 494824 | 494908 |
| 4 | 242587 | 485206 | 485356 |

**Table 4** Model selection of Poisson HMMs on hospital dataset

According to the AIC, the BIC, and the negative log-likelihood, the model with *m* = 4 is the most appropriate.

26 First Author*et al.* and dd: Running title

Once the 4 state HMM is estimated, we can look at the state distribution of a 4 state Poisson HMM fitted on the hourly arrivals.

3000

1

2

3

4

5

7

8

9

10

11 12 13

14

15

16

17 18 19 20

21

22

2000

amount

state

1

2

3

4

1000

0

0 6 23

1234 1234 1234 1234 1234 1234 123 1234 1234 1234 1234 1234 1234 1234 1234 1234 1234 1234 1234 1234 1234 1234 1234 123

hour

This graph shows the amount of states per hour across the entire dataset. The states (1, 2, 3, 4) have means (3.816, 8.64, 13.186, 22.503) respectively. This model seems to agree with our guess.

# Speed evaluation

## Lamb and simulation dataset description

We timed estimations using different parameters on a dataset provided by [Leroux and Puterman](#_bookmark32) ([1992](#_bookmark32)) and on a simulated dataset.

The first dataset is the number of movements by a fetal lamb in 240 consecutive 5-second intervals, and

is:

0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 2 2 0 0 0 0 1 0 0 1 1 0 0 1 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

2 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 7 3 2 3 2 4 0 0 0 0 1 0 0 0 0 0 0 0 1 0 2 0 0 0 0 1 0 0 0 0 1 0

0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 2 1 0 0 1 0 0 0 1 0 1 1 0 0 0 1 0 0 1 0 0 0 1 2 0 0 0 1 0 1 1 0 1 0 0 2 0 1 2

1 1 2 1 0 1 1 0 0 1 1 0 0 0 1 1 1 0 4 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0

**Table 5** Numbers of movements by a fetal lamb in 240 consecutive 5-second intervals

The second dataset is a sequence of random numbers generated by a *m*-state Poisson HMM. The code we used is to generate it is

Biometrical Journal **52** (2020) 61 27

DATA\_SIZE\_SIMUL <- 2000

m <- 2

*# Generate parameters*

lambda <- **seq**(1, 7, length.out = m)

*# Create the transition probability matrix with 0.8 on its diagonal*

**if** (m == 1) {

gamma <- **diag**(1, m)

} **else** {

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\_SIMUL,

mod = **list**(m = m,

lambda = lambda, gamma = gamma, delta = delta))

with *m* = 1*,* 2*,* 3.

## Speed comparison

1. Speed of TMB

To compare speeds with and without using TMB, it is important to test multiple sets of parameters. To do so, we used the R package microbenchmark.

The following [Figure 2](#_bookmark11) below shows the time acceleration when using TMB on the lamb dataset from [Leroux and Puterman](#_bookmark32) ([1992](#_bookmark32)) through boxplots.

28 First Author*et al.* and dd: Running title

Parameter estimation time

**Lamb data, size = 240, m = 2**

0.030

0.010

Time (seconds)

0.003

DM TMB1 TMB2 TMB3 TMB4

Exact/inexact gradient and hessian

**Figure 2** Time w/o using TMB with lamb data The average speed increases using the lamb dataset are summarized here:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| m | TMB1 | TMB2 | TMB3 | TMB4 |
| 1 | 2064 | 1922 | 2119 | 1825 |
| 2 | 2299 | 2261 | 4796 | 1657 |

**Table 6** Speed percentage increase of estimation using TMB on the lamb dataset

The following [Figure 3](#_bookmark12) below shows the time acceleration when using TMB on a simulated dataset through boxplots.

Biometrical Journal **52** (2020) 61 29

Parameter estimation time

**Simulated data, size = 2000**

1.00

0.10

Time (seconds)

0.01

|  |  |  |  |
| --- | --- | --- | --- |
| 1 |  | 2 | 3 |
| DM TMB1 TMB2 TMB3 | TMB4 | DM TMB1 TMB2 TMB3 TMB4 | DM TMB1 TMB2 TMB3 TMB4 |
|  |  | Exact/inexact gradient and hessian |  |

**Figure 3** Time w/o using TMB with simulated data The average speed increases using a simulated dataset are summarized here:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| m | TMB1 | TMB2 | TMB3 | TMB4 |
| 1 | 2093 | 2148 | 1935 | 1746 |
| 2 | 1757 | 1794 | 3536 | 3301 |
| 3 | 1125 | 1120 | 4528 | 2927 |

**Table 7** Speed percentage increase of estimation using TMB on the simulated dataset

30 First Author*et al.* and dd: Running title

1. Log-likelihoods

Optimizing with TMB gives the same estimates as optimizing without, as we can see in the following table when estimating a 2 state Poisson HMM on the lamb dataset.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | DM | TMB1 | TMB2 | TMB3 | TMB4 |
| *λ*1 | 0.26 | 0.26 | 0.26 | 0.26 | 0.26 |
| *λ*2 | 3.11 | 3.11 | 3.11 | 3.11 | 3.11 |
| *γ*1*,*1 | 0.99 | 0.99 | 0.99 | 0.99 | 0.99 |
| *γ*2*,*1 | 0.31 | 0.31 | 0.31 | 0.31 | 0.31 |
| *γ*1*,*2 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 |
| *γ*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** 2 state Poisson HMM estimates w/o TMB, lamb dataset

Moreover, the computation time of the likelihood is also smaller with TMB as we can see in the 3 following tables.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| m | TMB1 | TMB2 | TMB3 | TMB4 |
| 1 | 2063 | 2074 | 2103 | 2048 |
| 2 | 2004 | 2090 | 2124 | 2080 |

**Table 9** Speed percentage increase of TMB for negative log-likelihood calculation on the lamb dataset

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| m | TMB1 | TMB2 | TMB3 | TMB4 |
| 1 | 2097 | 2122 | 2138 | 2111 |
| 2 | 2279 | 2248 | 2307 | 2246 |
| 3 | 1133 | 1132 | 1101 | 1102 |

**Table 10** Speed percentage increase of TMB for negative log-likelihood calculation on the simulated dataset

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| m | TMB1 | TMB2 | TMB3 | TMB4 |
| 1 | 1477 | 1467 | 1482 | 1443 |
| 2 | 1620 | 1623 | 1617 | 1605 |
| 3 | 1012 | 1013 | 982 | 1000 |
| 4 | 683 | 704 | 699 | 701 |

**Table 11** Speed percentage increase of TMB for negative log-likelihood calculation on the hospital dataset

Biometrical Journal **52** (2020) 61 31

1. All optimization methods

Finally, we compared different optimization methods. The ones we retained were BFGS, Nelder- Mead, L-BFGS-B, nlm, nlminb, and hjn, because the others didn’t converge. We also added the algorithm for least-squares curve fitting from the package marqLevAlg. Exact gradients and hes- sians are provided by TMB and fed to each algorithm. The speed comparisons are in the following [Figure 4](#_bookmark14).

**Lamb data, size = 240, m = 2**

Parameter estimation time

**Simulated data, size = 2000, m = 3**

Parameter estimation time

5

0.5

3

Time (milliseconds)

Time (milliseconds)

0.3

2

0.2

BFGS L−BFGS−B nlm nlminb hjn marqLevAlg

Method

BFGS L−BFGS−B nlm nlminb hjn marqLevAlg

Exact/inexact gradient and hessian

**Figure 4** Log estimation time vs optimization method

32 First Author*et al.* and dd: Running title

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 show the lower and upper bounds of the confidence intervals. The reason is that sometimes, only one bound of the interval is known when using the profile likelihood method and doesn’t let us have a standard error.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| m | Parameter | Parameter.estimate | Profile.L | Profile.U | Bootstrap.L | Bootstrap.U | TMB.L | TMB.U |
| 1 | *λ*1 | 0.36 |  |  | 0.29 | 0.44 | 0.28 | 0.43 |
| 2 | *λ*1 | 0.26 | 0.15 | 0.33 | 0.00 | 0.33 | 0.18 | 0.34 |
| 2 | *λ*2 | 3.11 | 1.27 | 4.95 | 0.39 | 5.21 | 1.11 | 5.12 |
| 2 | *γ*1*,*1 | 0.99 | 0.93 | 1.00 | 0.56 | 1.00 | 0.97 | 1.01 |
| 2 | *γ*2*,*1 | 0.31 | 0.04 | 0.68 | 0.04 | 1.00 | -0.05 | 0.67 |
| 2 | *γ*1*,*2 | 0.01 | 0.00 | 0.07 | 0.00 | 0.44 | -0.01 | 0.03 |
| 2 | *γ*2*,*2 | 0.69 | 0.32 | 0.96 | 0.00 | 0.96 | 0.33 | 1.05 |
| 2 | *δ*1 | 0.96 |  |  | 0.96 | 0.96 | 0.90 | 1.03 |
| 2 | *δ*2 | 0.04 |  |  | 0.04 | 0.04 | -0.03 | 0.10 |

**Table 12** Estimates and standard errors on the lamb dataset

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| m | Parameter | Parameter.value | Parameter.estimate | Profile.L | Profile.U | Bootstrap.L | Bootstrap.U | TMB.L | TMB.U |
| 1 | *λ*1 | 1.00 | 0.97 |  |  | 0.93 | 1.01 | 0.93 | 1.02 |
| 2 | *λ*1 | 1.00 | 0.99 | 0.93 | 1.05 | 0.93 | 1.05 | 0.93 | 1.05 |
| 2 | *λ*2 | 20.00 | 20.12 | 19.83 | 20.40 | 19.83 | 20.43 | 19.83 | 20.40 |
| 2 | *γ*1*,*1 | 0.80 | 0.79 | 0.77 | 0.82 | 0.77 | 0.82 | 0.77 | 0.82 |
| 2 | *γ*2*,*1 | 0.20 | 0.22 | 0.20 | 0.25 | 0.20 | 0.25 | 0.20 | 0.25 |
| 2 | *γ*1*,*2 | 0.20 | 0.21 | 0.18 | 0.23 | 0.18 | 0.23 | 0.18 | 0.23 |
| 2 | *γ*2*,*2 | 0.80 | 0.78 | 0.75 | 0.80 | 0.75 | 0.80 | 0.75 | 0.80 |
| 2 | *δ*1 | 0.50 | 0.52 |  |  | 0.52 | 0.52 | 0.48 | 0.56 |
| 2 | *δ*2 | 0.50 | 0.48 |  |  | 0.48 | 0.48 | 0.44 | 0.52 |
| 3 | *λ*1 | 1.00 | 0.96 | 0.88 | 1.04 | 0.88 | 1.03 | 0.88 | 1.04 |
| 3 | *λ*2 | 10.50 | 10.30 | 10.03 | 10.58 | 10.04 | 10.58 | 10.03 | 10.58 |
| 3 | *λ*3 | 20.00 | 20.16 | 19.77 | 20.56 | 19.75 | 20.55 | 19.77 | 20.56 |
| 3 | *γ*1*,*1 | 0.80 | 0.81 | 0.77 | 0.85 | 0.78 | 0.84 | 0.78 | 0.84 |
| 3 | *γ*2*,*1 | 0.10 | 0.09 | 0.07 | 0.11 | 0.07 | 0.11 | 0.06 | 0.11 |
| 3 | *γ*3*,*1 | 0.10 | 0.10 | 0.08 | 0.12 | 0.07 | 0.12 | 0.07 | 0.12 |
| 3 | *γ*1*,*2 | 0.10 | 0.09 | 0.07 | 0.11 | 0.06 | 0.11 | 0.06 | 0.11 |
| 3 | *γ*2*,*2 | 0.80 | 0.83 | 0.79 | 0.87 | 0.80 | 0.86 | 0.80 | 0.86 |
| 3 | *γ*3*,*2 | 0.10 | 0.10 | 0.08 | 0.13 | 0.08 | 0.13 | 0.08 | 0.13 |
| 3 | *γ*1*,*3 | 0.10 | 0.10 | 0.08 | 0.13 | 0.08 | 0.13 | 0.08 | 0.13 |
| 3 | *γ*2*,*3 | 0.10 | 0.09 | 0.07 | 0.11 | 0.06 | 0.11 | 0.06 | 0.11 |
| 3 | *γ*3*,*3 | 0.80 | 0.80 | 0.75 | 0.84 | 0.77 | 0.83 | 0.77 | 0.83 |
| 3 | *δ*1 | 0.33 | 0.33 |  |  | 0.33 | 0.33 | 0.27 | 0.38 |
| 3 | *δ*2 | 0.33 | 0.35 |  |  | 0.35 | 0.35 | 0.30 | 0.41 |
| 3 | *δ*3 | 0.33 | 0.32 |  |  | 0.32 | 0.32 | 0.27 | 0.37 |

**Table 13** Estimates and standard errors on the simulated dataset

All the code used to produce those results is available in the supporting information. The files executing the code are poi hmm lamb.R, poi hmm simul.R, and poi hmm hosp.R.

# 8 Discussion

1. Throughout this paper, when we actually used hidden states (*m >* 1), we found that TMB3 yields the greatest speed increase (when TMB’s exact gradient is provided to the optimizer but not the hessian). This also stayed true when using other optimizers such as nlm. The reason is unclear. Although adding the exact hessian on top of the gradient to the optimizer seems to slow down the computation, it might help the optimizer converge in some cases, and could be worth investigating.
2. When providing only TMB’s hessian, optimizers sometimes fail to converge.
3. Mention the prospect of using TMB for panel data with random effects, Laplace approximation.

Biometrical Journal **52** (2020) 61 33

**Acknowledgements** We gratefully thank Dr. Bertrand GALICHON and Dr. Anthony CHAUVIN for their patience and their efforts to provide the hospital dataset along with the necessary authorizations.

## Conflict of Interest

*The authors have declared no conflict of interest.*

# Appendix

## R Code

1. Code to setup global parameters and declare functions used
2. Packages used
3. Functions used in TMB related code
4. Functions used in non TMB related code
5. Code to run estimations and comparisons using the lamb dataset
6. Code to run estimations and comparisons using a simulated dataset
7. Code to run estimations and comparisons using the hospital dataset

## C++ Code

1. Poisson HMM negative log-likelihood calculation This is the file poi hmm.cpp which contains the negative likelihood function.

It should be noted that in the C++ file, testing if the value is missing (i.e. testing for NaN (Not A Number) values) requires a little trick. The reason is that the standard test function std::isnan() doesn’t currently work on a single data value inside TMB.

Cuurently in C++, comparisons involving NaN values are always false, except when testing inequality between 2 NaN values. In other words, for a float f, the expression f != f will be true if and only if f is a NaN value. Similarly, f == f returns false if and only if f is a NaN value.

**# i n c l u d e** *<*TMB. hpp*>*

**# i n c l u d e** ” . . / f u n c t i o n s / u t i l s . cpp ”

*/ / L i k e l i h o o d f o r a poisson h i dden markov model .*

**template** *<***c l a s s** Type*>*

Type o b j e c t i v e f u n c t i o n *<*Type *>* : : **operator** ( ) ( )

*{*

*/ / Data*

DATA VECTOR( x ) ; DATA INTEGER(m) ;

*/ / t i m e s e r i e s v e c t o r*

*/ / Number o f s t a t e s m*

*/ / Parameters*

PARAMETER VECTOR( t l ambda ) ; PARAMETER VECTOR( tgamma ) ;

*/ / c o n d i t i o n a l l o g s d ' s*

*/ / m( m−1) working parameters o f TPM*

34 First Author*et al.* and dd: Running title

*/ / Uncomment only u s i ng a non s t a t i o n a r y d i s t r i b u t i o n*

*/ / PARAMETER VECTOR( t d e l t a ) ; / / t r ansformed s t a t i o n a r y d i s t r i b u t i o n ,*

*/ / Transform working parameters t o n a t u r a l parameters :*

v e c t o r *<*Type*>* lambda = t l ambda . exp ( ) ;

matrix *<*Type*>* gamma = Gamma w2n (m, tgamma ) ;

*/ / C o n s t r u c t s t a t i o n a r y d i s t r i b u t i o n*

v e c t o r *<*Type*>* d e l t a = S t a t d i s t (m, gamma ) ;

*/ / I f u s i ng a non s t a t i o n a r y d i s t r i b u t i o n , use t h i s i n s t e a d*

*/ / v ec t o r <Type> d e l t a = Delta w 2 n ( m, t d e l t a ) ;*

*/ / Get number o f t i m e s t e p s ( n )*

**i n t** n = x . s i z e ( ) ;

*/ /*

*/ /*

*/ /*

*Evaluate c o n d i t i o n a l d i s t r i b u t i o n : Put c o n d i t i o n a l*

*p r o b a b i l i t i e s o f observed x*

*( one column f o r each s t a t e ,* matrix *<*Type*>* e m i s s i o n p r o b s ( n , matrix *<*Type*>* row 1 vec ( 1 , m) ;

row 1 vec . s e t O n e s ( ) ;

*i n n t i m e s m matrix*

*one row f o r each d a t a p o i n t ) :*

m) ;

**f o r** ( **i n t** i = 0 ; i *<* n ; i ++) *{*

**i f** ( x [ i ] != x [ i ] ) *{ / / f != f r e t u r n s*

*t r u e i f and only i f*

*NaN i n C++) with 1*

*f i s NaN .*

*/ / Replace missing v a l u e s ( NA i n R ,*

e m i s s i o n p r o b s . row ( i ) = row 1 vec ;

*}*

**e l s e** *{*

e m i s s i o n p r o b s . row ( i ) = d p o i s ( x [ i ] , lambda , **f a l s e** ) ;

*}*

*}*

*/ / Corresponds t o t h e book page 333*

matrix *<*Type*>* foo , P ;

Type mllk , sumfoo , l s c a l e ;

**i f** (m == 1 ) *{*

mllk = *−* e m i s s i o n p r o b s . c o l ( 0 ) . a r r a y ( ) . l o g ( ) . sum ( ) ;

*/ / Use ad r e por t on v a r i a b l e s we are i n t e r e s t e d i n :*

ADREPORT( lambda ) ; ADREPORT( gamma ) ; ADREPORT( d e l t a ) ;

*/ / Things we need f o r l o c a l decoding*

REPORT( lambda ) ;

Biometrical Journal **52** (2020) 61 35

REPORT( d e l t a ) ;

**return** mllk ;

*}*

foo = ( d e l t a \* v e c t o r *<*Type *>*( e m i s s i o n p r o b s . row ( 0 ) ) ) . m a t r i x ( ) ; sumfoo = foo . sum ( ) ;

l s c a l e = l o g ( sumfoo ) ;

foo . t r a n s p o s e I n P l a c e ( ) ; foo /= sumfoo ;

**f o r** ( **i n t** i = 2 ; i *<*= n ; i ++) *{*

P = e m i s s i o n p r o b s . row ( i *−* 1 ) ;

foo = ( ( foo \* gamma ) . a r r a y ( ) \* P . a r r a y ( ) ) . m a t r i x ( ) ; sumfoo = foo . sum ( ) ;

l s c a l e += l o g ( sumfoo ) ; foo /= sumfoo ;

*}*

mllk = *−* l s c a l e ;

*/ / Use ad r e po r t on v a r i a b l e s f o r which we want s t andard e r r o r s*

ADREPORT( lambda ) ; ADREPORT( gamma ) ; ADREPORT( d e l t a ) ;

*/ / V a r i a b l e s we need f o r l o c a l decoding and conveniency*

REPORT( lambda ) ; REPORT( gamma ) ; REPORT( d e l t a ) ; REPORT( n ) ;

REPORT( e m i s s i o n p r o b s ) ; REPORT( mllk ) ;

**return** mllk ;

*}*

1. This optional function is in the file utils.cpp. It is only necessary if a stationary distribution is not assumed, and ***δ*** is used as a parameter instead of being derived from the transistion probability matrix **Γ**. It codes the function to convert the working parameter tdelta into its natural format.

*/ / Function t r a n s f o r m i n g working parameters i n i n i t i a l d i s t r i b u t i o n*

*/ / t o n a t u r a l parameters*

**template** *<***c l a s s** Type*>*

v e c t o r *<*Type*>* Delta w 2 n ( **i n t** m, v e c t o r *<*Type*>* t d e l t a ) *{*

v e c t o r *<*Type*>* d e l t a (m) ; v e c t o r *<*Type*>* foo (m) ;

**i f** (m == 1 )

36 First Author*et al.* and dd: Running title

**return** Type ( 1 ) ;

*/ / s e t f i r s t e l ement t o one .*

*/ / F i l l i n t h e l a s t m − 1 e l em en t s with working parameters*

*/ / and t a k e e x p o n e n t i a l*

foo *<<* Type ( 1 ) , t d e l t a . exp ( ) ;

*/ / normalize*

d e l t a = foo / foo . sum ( ) ;

**return** d e l t a ;

*}*

1. This function is in the file utils.cpp. It transforms the transition probability matrix **Γ** from its working format to its natural format.

*/ / Function t r a n s f o r m i n g t h e working parameters i n TPM t o*

*/ / n a t u r a l parameters ( w2n )*

**template** *<***c l a s s** Type*>*

matrix *<*Type*>* Gamma w2n ( **i n t** m, v e c t o r *<*Type*>* tgamma ) *{*

*/ / C o n s t r u c t m x m i d e n t i t y matrix*

matrix *<*Type*>* gamma (m, m) ; gamma . s e t I d e n t i t y ( ) ;

**i f** (m == 1 )

**return** gamma ;

*/ / F i l l o f f d i a g o n a l e l e me n t s with working parameters column−wise :*

**i n t** i d x = 0 ;

**f o r** ( **i n t** i = 0 ; i *<* m; i ++)*{*

**f o r** ( **i n t** j = 0 ; j *<* m; j ++)*{*

**i f** ( j != i ) *{*

*/ / F i l l gamma according t o mapping and t a k e e x p o n e n t i a l*

gamma ( j , i ) = tgamma . exp ( ) ( i d x ) ; i d x ++;

*}*

*}*

*}*

*/ / Normalize each row :*

v e c t o r *<*Type*>* cs = gamma . rowwise ( ) . sum ( ) ;

**f o r** ( **i n t** i = 0 ; i *<* m; i ++) gamma . row ( i ) /= cs [ i ] ;

**return** gamma ;

*}*

Biometrical Journal **52** (2020) 61 37

1. This function is in the file utils.cpp. It derives the stationary distribution from the transition probability matrix **Γ**.

*/ / Function computing t h e s t a t i o n a r y d i s t r i b u t i o n o f a Markov chain*

**template** *<***c l a s s** Type*>*

v e c t o r *<*Type*>* S t a t d i s t ( **i n t** m, matrix *<*Type*>* gamma ) *{*

*/ / C o n s t r u c t s t a t i o n a r y d i s t r i b u t i o n*

matrix *<*Type*>* I (m, m) ; matrix *<*Type*>* U(m, m) ;

matrix *<*Type*>* row 1 vec ( 1 , m) ; U = U. s e t O n e s ( ) ;

I = I . s e t I d e n t i t y ( ) ; row 1 vec . s e t O n e s ( ) ;

matrix *<*Type*>* A = I *−* gamma + U;

matrix *<*Type*>*

matrix *<*Type*>*

v e c t o r *<*Type*>*

Ainv = A. i n v e r s e ( ) ;

d e l t a m a t = row 1 vec

\*

Ainv ;

d e l t a = d e l t a m a t . row ( 0 ) ;

**return** d e l t a ;

*}*

1. Functions used in C++ Poisson HMM code
2. Linear model negative log-likelihood calculation
3. Functions used in C++ linear model code
4. simple C acceleration of Zucchini scripts p. 333, A 1.7, A 1.8 with conditional probabilities outside of the forward / backward loop
5. Use same order as in Zucchini
6. the .cpp file with transformation code
7. the .cpp file with likelihood
8. an .R file showcasing the use

38 First Author*et al.* and dd: Running title

# 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, 1966. ISSN 00034851.

Olivier Cappe´, Eric Moulines, and Tobias Ryden. *Inference in Hidden Markov Models*. Springer Science & Business Media, April 2006. ISBN 978-0-387-28982-3.

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 Fru¨hwirth-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 Ha¨rdle, 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 Mod- els (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.

Eric E. Schadt, Janet S. Sinsheimer, and Kenneth Lange. Computational Advances in Maximum Likelihood Methods for Molecular Phylogeny. *Genome Research*, 8(3):222–233, March 1998. ISSN 1088-9051, 1549-5469. doi: 10.1101/gr.8.3.222.

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.