Non-homogeneous Markov and misclassification hidden Markov multi-state modelling in R

Andrew C. Titman

School of Mathematical Sciences, Lancaster University, UK

August 31, 2025

Abstract

Multi-state models are a useful approach for modelling event history data in which an individual makes transitions between a series of states over time. A Markov assumption is often made in such models and for intermittently observed data the models are usually parametric.

The *nhm* package allows non-homogeneous Markov models with smoothly changing transition intensities to be fitted to intermittently observed data through direct numerical solution of the differential equations defining the transition probabilities. Misclassification type hidden Markov models with non-homogeneous transition intensities can also be fitted. The package permits models with log-linear time trends (Gompertz type intensities) to be fitted, Weibull intensities or B-spline intensity functions to be specified directly. In addition, users may also supply their own function for the generator function of transition intensities and its derivatives to allow bespoke models to be fitted.

This manual provides a brief overview of the underlying theory behind non-homogeneous Markov and misclassification hidden Markov models and also gives a tutorial on the typical use of *nhm*.

1 Introduction

The likelihood for Markov models on intermittently observed data requires computation of the transition probabilities, which are the solution of the Kolmogorov Forward Equations (KFE) a system of ordinary differential equations defined by the transition intensities [2]. For time homogeneous models where the intensities are constant, the KFE define a linear system of equations and hence the transition probabilities can be calculated by computing a matrix exponential. Moreover, the transition probabilities for models with piecewise constant intensities can also be found

in matrix analytic form by computing matrix exponentials with respect to each time period in which the intensities are constant and combining them using the Chapman-Kolmogorov equations. Models using these matrix analytic formulations can be fitted efficiently using the *msm* package [6].

However, the assumption of time homogeneity is often not plausible and models with piecewise constant intensities are potentially sensitive to the choice of cut points.

2 Underlying methods

The underlying methods implemented by the package overlap heavily with those of the *msm* package [6] and the reader is encouraged to consult the *msm* package manual and references therein for a basic introduction to intermittently observed time homogeneous multi-state models. See in particular, Kalbfleisch and Lawless (1985) [8] and Jackson *et al* (2003) [7] for the seminal papers for Markov and misclassification-type Markov models, respectively.

The key extension that *nhm* aims to accommodate is the fitting of models where the transition intensities may be smooth functions of time.

The likelihood for an individual observed in states $x_0, x_1, \ldots, x_m$ at times $t_0, t_1, \ldots, t_m$ can be expressed as

$$L(\boldsymbol{\theta}) = \prod_{j=1}^{m} \mathbb{P}(X(t_j) = x_j | X(t_{j-1}) = x_{j-1}; \boldsymbol{\theta}) = \prod_{j=1}^{m} p_{x_{j-1}x_j}(t_{j-1}, t_j; \boldsymbol{\theta})$$

Let $\mathbf{P}(t_0, t; \boldsymbol{\theta})$ be the matrix of transition probabilities such that $\{\mathbf{P}(t_0, t)\}_{rs} = p_{rs}(t_0, t; \boldsymbol{\theta})$ then for a general non-homogeneous Markov model, these transition probabilities satisfy the initial value problem

$$\frac{d\mathbf{P}(t_0, t; \boldsymbol{\theta})}{dt} = \mathbf{P}(t_0, t; \boldsymbol{\theta})\mathbf{Q}(t; \boldsymbol{\theta}), \;\; \mathbf{P}(t_0, t_0; \boldsymbol{\theta}) = \mathbf{I}$$

where $\mathbf{Q}(t; \boldsymbol{\theta})$ is the generator matrix of transition intensities. Following the approach proposed in Titman (2011), *nhm* uses direct numerical solution of this system of differential equations to compute the likelihood. For additional efficiency in optmization, Titman (2011) proposed to solve the extended system of differential equations, incorporating the systems of equations defining the derivatives with respect to the parameter vector $\boldsymbol{\theta}$:

$$\frac{d\mathbf{P}'(t_0, t; \boldsymbol{\theta})}{dt} = \mathbf{P}'(t_0, t; \boldsymbol{\theta})\mathbf{Q}(t; \boldsymbol{\theta}) + \mathbf{P}(t_0, t; \boldsymbol{\theta})\mathbf{Q}'(t; \boldsymbol{\theta}), \;\; \mathbf{P}'(t_0, t_0; \boldsymbol{\theta}) = \mathbf{0}$$

where

$$\mathbf{P}'(t_0, t_0; \boldsymbol{\theta}) = \frac{\partial \mathbf{P}(t_0, t; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \text{ and } \mathbf{Q}'(t; \boldsymbol{\theta}) = \frac{\partial \mathbf{Q}(t; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}},$$

so that the first derivatives of the likelihood can be calculated.

The differential equations are solved by using the LSODA routine [11] of the deSolve package [12], which automatically selects appropriate methods to allow for *stiff* differential equations, if necessary.

2.1 Misclassification type hidden Markov models

For misclassification type hidden Markov models, forward recursion is used to compute the likelihood contribution for an individual.

Let $e_{rs} = P(O_j = s|X_j = r)$ be the (r, s) entry of the misclassification matrix. Then for a subject with observed states $O_1, \ldots, O_m$ at times $t_1, \ldots, t_m$, forward weights are defined as $\alpha_k(j)$ for observation number $k = 1, \ldots, m$ and state $j = 1, \ldots, R$, where

$$\alpha_1(j) = \mathbb{P}(O_1, X_1 = j) = \pi_{0j} e_{j,O_1},$$

where π_{0j} is the jth entry of $\pi_{\mathbf{0}}$, and subsequent forward weights are calculated recursively:

$$\alpha_k(j) = \mathbb{P}(O_1, \ldots, O_k, X_k = j) = \sum_{i=1}^{R} \alpha_{k-1}(i) e_{j,O_k} p_{ij}(t_{k-1}, t_k; \boldsymbol{\theta}).$$

Then the likelihood for that subject is given by

$$\mathbb{P}(O_1, \ldots, O_m) = \sum_{i=1}^{R} \alpha_m(i).$$

To compute the first derivatives, an extended forward recursion is used as proposed by Lystig and Hughes (2002) [9] in the context of discrete time hidden Markov models.

If we further define

$$\phi_k(\theta_u, j) = \frac{\partial \alpha_k(j)}{\partial \theta_u} = \frac{\partial}{\partial \theta_u} \mathbb{P}(O_1, \ldots, O_k, X_k = j)$$

then this allows $\phi_k(\theta_u, j)$ to be calculated recursively as

$$\begin{aligned} \phi_k(\theta_u, j) = \sum_{i=1}^{R} \Big(& \phi_{k-1}(\theta_u, i) e_{jO_k} p_{ij}(t_{k-1}, t_k; \boldsymbol{\theta}) + \\ & \alpha_{k-1}(i) \frac{\partial e_{j,O_k}}{\partial \theta_u} + \alpha_{k-1}(i) e_{j,O_k} \frac{\partial p_{ij}(t_{k-1}, t_k; \boldsymbol{\theta})}{\partial \theta_u} \Big). \end{aligned} \tag{2.1}$$

Then the first derivative of the likelihood for the subject is given as

$$\frac{\partial \mathbb{P}(O_1, \ldots, O_m)}{\partial \theta_u} = \sum_{i=1}^{R} \phi_m(\theta_u, i).$$

3 Model specification: *model.nhm*

The purpose of the function `model.nhm` is to define model objects which may then be used in the main function, `nhm`. `model.nhm` puts the supplied data into a standardized format, builds functions for computing the generator or intensity matrix and its derivatives and, in the case of misclassification models, also creates functions for computing the misclassification probabilities and initial state occupation probabilities.

3.1 Identification of key variables

The `formula` argument follows the same syntax as used in the **msm** package. A formula should be provided where the left hand side identifies the state variable and the right hand side identifies the time variable. For example

`formula = state ~ years` implies that the observed states are stored in a column named `state` while the corresponding observation times are in a column `years`.

The `data` argument identifies the name of the data frame in which the data are stored. Note that this cannot be omitted. The `subject` argument identifies the name of the subject identifier variable within the data frame. The `covariates` argument should be a character vector giving the names of the covariates that are used in the model. These variables should be columns within the `data` data frame. Note that these covariates could be used in either the model for the transition intensities or, in the case of misclassification models, for covariate effects on the misclassification probabilities or initial state probabilities.

3.2 Generator matrix *type*

The `type` argument specifies the type of non-homogeneous model for the generator or intensity matrix of the Markov process. The possible values are `'gompertz'`, `'weibull'`, `'bspline'` and `'bespoke'`.

Gompertz type

A `'gompertz'` type model leads to models where some or all of the intensities are of the form

$$q_{rs}(t;\mathbf{z}) = \exp(\theta_{rs} + \beta_{rs}(t-\bar{t}) + \boldsymbol{\gamma}_{rs}^{'}\mathbf{z})$$

where β_{rs} defines a log-linear trend in intensity with respect to time, with the model reducing to time homogeneity if $\beta_{rs} = 0$. Here, $\bar{t}$ is an optional centring term which reparametrizes θ_{rs} to refer to the baseline intensity value at $t = \bar{t}$ rather than 0. Such centring can be specified using the optional argument `centre_time` and usually helps with convergence of the maximum likelihood algorithm.

Weibull type

A `'weibull'` type model leads to models where some or all of the intensities are of the form

$$q_{rs}(t;\mathbf{z}) = \lambda_{rs}\alpha_{rs}(\lambda_{rs}t)^{\alpha_{rs}-1}\exp(\boldsymbol{\gamma}_{rs}^{'}\mathbf{z})$$

where $\alpha_{rs} > 0$ is the shape parameter, $\lambda_{rs} > 0$ the rate parameter. A time homogeneous model arises with $\alpha_{rs} = 1$. In *nhm*, in order to allow the likelihood to be maximized via unconstrained optimization, the model is parametrized in terms of the log-shape, a_{rs}, and log-rate, θ_{rs},

$$q_{rs}(t;\mathbf{z}) = \exp\left\{\theta_{rs}\exp(a_{rs}) + (\exp(a_{rs})-1)\log t + \boldsymbol{\gamma}_{rs}^{'}\mathbf{z}\right\}.$$

B-spline type

A `'bspline'` type model leads to models where some or all of the intensities are of the form

$$q_{rs}(t; \mathbf{z}) = \exp\left\{\theta_{rs} + \boldsymbol{\nu}_{rs}^{'}\mathbf{B}(t) + \boldsymbol{\gamma}_{rs}^{'}\mathbf{z}\right\}$$

where $\mathbf{B}(t)$ is the vector of B-spline basis functions at time t for a B-spline of a certain order with certain knot points, implying the overall effect of time is represented by a spline function that is a smooth piecewise polynomial. Note that this specification differs slightly from that used in Titman (2011).

The knot points for the spline are specified using the argument `splinelist`. This should be a list of length equal to the number of distinct spline effects in the model, of vectors which specify the location of the knots of the spline (including the boundary knots). By default the splines are of degree 3, but this can be modified by using the `degrees` argument.

Bespoke type

If a non-homogeneous model not built into the package is desired, the `'bespoke'` type can be specified. To fit a bespoke model, it is necessary to also provide a function that computes the generator (transition intensity) matrix and its derivatives at a given time `t`, covariate vector `z` and vector of model parameters `x`. It should output a list containing a matrix `q` and an array `qp`. The matrix `q` should be the $R \times R$ matrix $\mathbf{Q}(t; z, \boldsymbol{\theta})$ whose (r, s) entry is $q_{rs}(t; z, \boldsymbol{\theta})$ for $r \neq s$ and $-\sum_k q_{rk}(t; z, \boldsymbol{\theta})$ for $r = s$. The array `qp` should be the $R \times R \times n_p$ array of first derivatives $d\mathbf{Q}(t; z, \boldsymbol{\theta})/d\boldsymbol{\theta}$, where n_p is the number of parameters in the model for the generator matrix.

Although not required, it is desirable to provide an attribute to the function named `npar` that stores n_p. In addition, parameter names (for use in `print`) can be supplied in an attribute `parnames`. The attribute should be a character string of length n_p. The created function should be supplied as the argument `intens`.

Note that for large models or datasets or if the same model will be fitted many times, it is general advantageous to hard code an `intens` function even if the model is one of the pre-specified types. The speed gain varies, but may be biggest for models with a large(r) number of covariates that only affect a small number of transitions.

3.3 *trans* argument

The number of states in the model and the set of admissible transitions is specified using the `trans` argument. `trans` should be an $R \times R$ matrix. The (r, s) entry should be 0 if the $r \to s$ transition is inadmissible. The admissible transitions should be numbered consecutively from 1. If the same number is given to two or more distinct transitions, e.g. (r, s) and (r', s'), then this implies $\theta_{rs} = \theta_{r's'}$, i.e. the baseline parameters for each transition are constrained to be equal.

The diagonal of the matrix will be ignored, but a warning is given if it contains entries other than 0.

For `bespoke` type models, `trans` is only used to specify the number of states in the model, i.e. it can be an $R \times R$ matrix of zeroes.

3.4 *nonh* argument

The `nonh` argument specifies which transition intensities are non-homogeneous with respect to time. The same principles as the verb!trans! argument holds. `trans` should be an $R \times R$ matrix. If the (r, s) entry is 0 it implies either the transition is inadmissible or it is time homogeneous. The non-homogeneous transitions should be numbered consecutively from 1. If the same number is given to two or more transitions then it is assumed they have the same time trend parameter(s). For instance for a Weibull type model it would imply that $\alpha_{rs} = \alpha_{r's'}$.

3.5 *covm* argument

The `covm` argument is used to specify which transitions are affected by which covariates. The argument can either be supplied as a named list of $R \times R$ matrices or else an $R \times R \times n_c$ array, where n_c is the length of the `covariates` argument.

If a named list is supplied, the names must be present in the `covariates` vector. If an array is supplied, the (r, s, k) entry will relate to the effect of the *k*th covariate named in the `covariates` vector on the $r \to s$ transition. The same principle used in `trans` and `nonh` is used for `covm`. A zero entry implies no effect, otherwise effects should be numbered consecutively from 1. Assigning the same number to any two entries implies a common covariate effect. **NB:** The numbering applies across the whole argument.

3.5.1 Time dependent covariates

Time dependent covariates may be included in the model. Time dependent covariates are assumed to be constant between observed visit times. The same data convention as the *msm* package is used, namely that the transition intensities for the period (t_i, t_{i+1}), will be governed by the covariates measured at t_i. As a consequence the covariates supplied for the last follow-up time will only be used in the case of exact death times (or for covariates on misclassification probabilities). Note that a particular advantage of *nhm* is that deterministic time dependent covariates can be incorporated into the model without using a piecewise constant approximation. For instance, if current age is to be a covariate such that

$$\lambda_{rs}(t; \text{age}(t)) = \exp(\theta_{rs} + \beta_{rs}\text{age}(t))$$

then this can be fitted via a “Gompertz" model

$$\lambda_{rs}(t; \text{age}(t)) = \exp(\theta_{rs} + \beta_{rs}(\text{age}_0 + t))$$

where age_0 is the patient's age at time 0. Note that the `centre_time` argument can be a vector, so supplying the vector of $-\text{age}_0$ will allow this model to be fitted. More complicated time dependent covariates can be accommodated

using a `bespoke` type intensity function.

3.6 Example: Gompertz model with covariate effects

We consider a four state progressive model where the generator matrix is of the form

$$\mathbf{Q}(t;\mathbf{z}) = \begin{bmatrix} -q_{12}(t;\mathbf{z}) & q_{12}(t;\mathbf{z}) & 0 & 0 \\ 0 & -q_{23}(t;\mathbf{z}) & q_{23}(t;\mathbf{z}) & 0 \\ 0 & 0 & -q_{34}(t;\mathbf{z}) & q_{34}(t;\mathbf{z}) \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

To specify that $1 \to 2$, $2 \to 3$ and $3 \to 4$ are the only admissible transitions and that they have distinct baseline intensity parameters:

```
trans <- rbind(c(0,1,0,0),c(0,0,2,0),c(0,0,0,3),rep(0,4))
```

We desire a model where all transitions have distinct time effects

```
nonh <- rbind(c(0,1,0,0),c(0,0,2,0),c(0,0,0,3),rep(0,4))
```

and that each of the covariates affects the $1 \to 2$ only

```
covm <- list(
cov1 = rbind(c(0,1,0,0),rep(0,4),rep(0,4),rep(0,4)),
cov2 = rbind(c(0,2,0,0),rep(0,4),rep(0,4),rep(0,4)))
```

The call to `model.nhm` is then

```
gomp_model <- model.nhm(state~time, data=example_data1, subject = id, covariates=c("cov1",
"cov2"),type="gompertz",trans=trans,nonh=nonh,covm=covm)
```

3.7 Arguments for misclassification models

The arguments `emat`, `ecovm`, `firstobs`, `initp`, `initp_value` and `initcovm` are specific to misclassification type hidden Markov models.

3.7.1 Misclassification probabilities

The general model for the misclassification probabilities is of the form

$$P(O_j = s | X_j = r, \mathbf{z}, \boldsymbol{\theta}) = \frac{\exp(u_{rs})}{\sum_k \exp(u_{rk})}$$

where $u_{rr} \equiv 0$ and $u_{rs} = \eta_{rs} + \boldsymbol{\tau}_{rs}^{'}\mathbf{z}$. Hence a multinomial logistic regression model is used for each state, with correct classification taken as the baseline.

The `emat` argument follows the same principles as `trans` in that it should be an $R \times R$ matrix numbered consecutively from 1. Inadmissible misclassification is labelled 0, while assigning the same number to two transitions implies that $\eta_{rs} = \eta_{r's'}$. `ecovm` follows the same principles as `covm` in defining the covariate effects $\boldsymbol{\tau}$.

3.7.2 Initial state probabilities

For Markov models without misclassification, the likelihood is computed conditional on the first observed state. In contrast, for misclassification models there are several different possible ways the process could be started. The purpose of the `firstobs` argument is to identify which of these possibilities applies.

The *firstobs* argument

Specifying `firstobs="exact"` implies that the state occupied at the first observation time was known without misclassification for all patients. This is the default if `firstobs` is not specified.

Specifying `firstobs="absent"` implies that no state was actually observed at the first observation time for each patient. Instead all that is assumed is the process was initiated at that time, with the state occupation probabilities at that time based on the model for initial probabilities. Note that the values of the first observations given in the `state` variable in this case are effectively place-holders required only to specify the time at which the patient's process was initiated.

Specifying `firstobs="misc"` implies that the state observed at the first observation time was subject to misclassification, with the distribution of true occupied states at that time based on the model for initial probabilities.

Model for initial probabilities

If `firstobs` is taken to be either `'absent'` or `'misc'` then a model for the state occupation probabilities at the first observation time is needed. By default, if no other arguments are specified it is assumed that the subject is in state 1 with probability 1.

One can either specify a fixed vector of initial probabilities using the `initial_value` argument or the initial probabilities can be allowed to follow a multinomial logistic regression model with state 1 taken as the baseline category and the parameters assumed unknown and to be estimated. In the former case `initial_value` should be a numerical vector of length R of non-negative values which sum to 1, corresponding to $P(X(t_0) = r), r = 1, \ldots, R$. In the latter case the argument `initp` should be a vector of length R which identifies which states other than 1 have a non-zero probability of being occupied at the initial time. As above, the states should be numbered consecutively from 1. If the same number is assigned twice or more it implies the same baseline parameter is used. The first entry of the vector is ignored, but expected to be 0. In addition `initpcovm` can be supplied to allow covariate effects on the initial probabilities. This should either be a named list of vectors of length R or an $R \times n_c$ matrix.

3.8 Further arguments

3.8.1 'Exact' death times

When a multi-state model describes life history data and absorbing states correspond to death or different causes of death, it is common for the time of death to be known up to the exact day. Treating the death time as if it were a discrete event time leads to bias. To ensure the correct likelihood contribution, `death = TRUE` should be specified and `death.states` should be a vector identifying which of the states is subject to exact death times. Note `death.states` should correspond to absorbing states within the data.

3.8.2 Censoring

For life history data, the end of follow-up for death may not correspond to the last observation time. For instance, the last observation of a patient may be at 5 years of follow-up, but the study ends at 7 years of follow-up. While we do not know the state occupied at 7 years, the absence of a death time can be assumed to be evidence that the patient is still alive. As such the patient's state occupied at 7 years is censored within the set of living states.

To accommodate this situation the supplied data should include the times of end of follow up for censored individuals with a corresponding censoring code or codes. The argument `censor` should be a vector of numerical censoring codes, while `censor.states` should be a list of vectors specifying the corresponding sets of possible states implied by the censoring codes. For instance `censor=c(98,99)` and `censor.states=list(1:2,1:3)` specifies that `98` implies the individual is in either state 1 or 2, while `99` implies the individual is in either state 1, 2 or 3. If `censor` is a single value then `censor.states` can be specified as a vector.

For models without misclassification, censoring can only occur as the last observation in a patient's sequence. Since version 0.1.2, misclassification models can accommodate censoring at any position within a patient's sequence. Note that a model without explicit misclassification can be treated as one by supplying an `emat` argument that is simply an $R \times R$ matrix of zeroes (implying the misclassification matrix is always the identity matrix).

3.9 Output of *model.nhm*

The `model.nhm` function outputs an object of class `nhm_model`. The print method for this object prints some basic information on the model specified; the type of model fitted, the number of unknown parameters and a table describing the individual parameters in the model. It is useful to check the output of the model specification to ensure it is as expected. It can also help when specifying initial parameter values in the `nhm` function.

The table of parameters includes a 'type' column. For non-bespoke generator matrix models this identifies which parameters correspond to baseline parameters (`Trans` - θ_{rs}), which parameters correspond to those defining non-homogeneity (`Nonhom` - β_{rs}, a_{rs} or ν_{rs}), which to the covariate effects on the intensities (`Cov` - $\boldsymbol{\gamma}_{rs}$), which to parameters for the misclassification model (`Emat`) and which to the initial state probabilities model (`Initp`). For

models of bespoke type, all the parameters relating to the generator matrix are taken to be of `Bespoke` parameter type. The identification of the `Nonhom` type parameters is important for the default functioning of the score test options (see Section 4.4)

For the example in Section 3.6, the output is as follows:

```
gomp_model

nhm model object for a Markov model.

Model for intensities has 8 unknown parameters and is of gompertz type.

                     Name   Type
1              Base: 1->2  Trans
2              Base: 2->3  Trans
3              Base: 3->4  Trans
4             Trend: 1->2 Nonhom
5             Trend: 2->3 Nonhom
6             Trend: 3->4 Nonhom
7 Covariate: cov1 1 -> 2    Cov
8 Covariate: cov2 1 -> 2    Cov
```

4 Model fitting: *nhm*

Once a model object has been created using `model.nhm` it can be passed to the main function `nhm`, which allows the likelihood to be computed and maximized.

The arguments of `nhm` control the way in which the model is maximized and which model outputs are required.

The `model_object` argument is for the `nhm_model` object created using `model.nhm`.

4.1 Specification of initial parameter values

The `initial` argument should correspond to the vector of initial parameter values. This should be a numeric vector of length equal to the number of unknown parameters in the model. For models without misclassification `initial` can be omitted and `gen_inits=TRUE` may be specified. In that case the initial parameter values are specified by calling the function `crudeinits.msm` from the **msm** package. Specifically initial parameters for the baseline transition intensities are found by assuming a homogeneous Markov model and that the observation times correspond to the exact transition times. Covariate effects and any non-homogeneity parameters are all set to zero. This method is not available for models with misclassification.

Note that the choice of initial parameter values can be very important to the eventual success of the optimization routine. Poor initial values may lead either to the optimization routine taking much longer to converge, or could lead to non-convergence. It can also cause problems for the computation of the transition probabilities (see [...] below).

It is best to build the model incrementally. A good starting point is usually a time homogeneous model. A time homogeneous version of the model can be fitted using **msm** and those parameters can be used as starting values. Unfortunately, due to the differences in syntax and the wider range of options in **msm** this process is not currently automated.

4.2 Control options

The `nhm` function contains a `control` argument. The function `nhm.control` should be used to create the list of control options.

4.2.1 Options for *deSolve*

The control options `tmax`, `rtol` and `atol` relate to parameters to be passed to the `lsoda` function within *deSolve*. The option `tmax` determines the maximum point at which the system of differential equations should be solve, while `rtol` and `atol` specify the relative and absolute tolerance levels it should use.

4.2.2 Algorithm options

In general the likelihood is optimized by using Newton-type algorithms in which the parameters are updated in the form

$$\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} - \delta^{(i)} \tilde{\mathbf{H}}(\boldsymbol{\theta}^{(i)})^{-1} \mathbf{u}(\boldsymbol{\theta}^{(i)})$$

where $\mathbf{u}(\boldsymbol{\theta}) = \frac{\partial \log L(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$, $\tilde{\mathbf{H}}(\boldsymbol{\theta}^{(i)})$ is some estimate of the Hessian and $\delta^{(i)}$ is some scalar, usually in $[0, 1]$, that determines the step length.

BHHH algorithm

The default optimization method used by `nhm` is the BHHH algorithm [1], implemented through the `maxLik` package [5]. The BHHH algorithm exploits the identity $\mathbb{E}(\mathbf{U}\mathbf{U}^{'}) = \mathbb{E}(\mathbf{I})$. This is a quasi-Newton algorithm in which the Hessian is approximated by $-\mathbf{J}(\boldsymbol{\theta}^{(i)})$ where

$$\mathbf{J}(\boldsymbol{\theta}^{(i)}) = \sum_{i=1}^{N} \mathbf{u}_i(\boldsymbol{\theta}^{(i)}) \mathbf{u}_i(\boldsymbol{\theta}^{(i)})^{'}$$

where $\mathbf{u}_i(\boldsymbol{\theta}^{(i)}) = \frac{\partial log L_i(\boldsymbol{\theta}^{(i)})}{\partial \boldsymbol{\theta}}$ and $L_i(\boldsymbol{\theta}^{(i)})$ is the likelihood contribution for subject i. If taking $\delta^{(i)} = 1$ does not result in an improvement to the likelihood, the algorithm will repeatedly halve the step length. The degree of detail of the progress of the BHHH algorithm that is printed can be controlled using `print.level` which should be an integer between 0 and 3. Further control options of the algorithm can be passed to *maxLik* using `maxLikcontrol`.

Fisher scoring

For models without misclassification, censoring or exact death times, a Fisher scoring algorithm can instead be used by specifying `fishscore=TRUE`. This uses minus the expected Fisher information as the estimate of the Hessian. For models that are well identified and are supplied good starting values, the Fisher scoring algorithm tends to be quicker. However, by default the algorithm is not very robust and may fail if the log-likelihood surface is quite flat or if a poor set of initial parameters is supplied. There are some additional options available to improve the robustness; `linesearch` allows a line search to be performed at each iteration to find the best step length, $\delta^{(i)}$. For models that are close to non-identifiable, the algorithm can be modified so that a diagonal term is added to the expected Fisher information, i.e. $\tilde{\mathbf{H}}(\boldsymbol{\theta}^{(i)}) = -\left\{\mathbb{E}(\mathbf{I}(\boldsymbol{\theta}^{(i)})) + \lambda \mathbf{I}_p\right\}$ where $\mathbf{I}_p$ is a $p \times p$ identity matrix. This modification can be specified by setting `damped=TRUE` and the damping parameter λ can be specified using `damppar`.

By default, for either the BHHH or Fisher scoring algorithms, a finite differences estimate of the observed Fisher information, $-\frac{\partial^2 \log L}{\partial \boldsymbol{\theta} \partial \boldsymbol{\theta}'}$, is computed. This requires an additional $2p$ evaluations of the likelihood gradient. While the observed information is generally considered to give more accurate standard error estimates, if `obsinfo=FALSE` then `nhm` will instead only provide either the expected Fisher information (in the case of the Fisher scoring algorithm) or $\mathbf{J}(\hat{\boldsymbol{\theta}})$ (in the case of the BHHH algorithm).

Constrained optimization

If the option `constrained` is set to `TRUE` then the NLMINB algorithm [4], implemented via `nlminb()` in the `stats` package, is used for optimization. The square of the individual score contributions is supplied to the `hessian` argument (rather than the exact Hessian), which leads to similar behaviour to the BHHH algorithm. However, `nlminb` allows lower and upper bounds to be specified for each of the parameters to be optimized. The main practical reason to use constrained optimisation is in order to make the algorithm more robust. Particularly for datasets with long follow-up periods and/or for models with Gompertz-type (log-linear) transition intensities, unless very good starting values are supplied there is significant potential for other algorithms to propose parameter values which lead to either numerical under/overflow or a zero value for the likelihood (see also Section 4.2.3). The `lower` and `upper` arguments can be used to force each parameter to stay within some plausible range, where the aim is to ensure that the magnitude of the resulting transition intensities cannot get either too high or too low to lead to under/overflow problems. Potentially, this process may need to be iterative: if the algorithm still fails it may be necessary to enforce stricter limits, whilst if the algorithm converges with one of the parameters at its lower or upper boundary, it would be necessary to restart with less stringent bounds.

4.2.3 Splits

By default `nhm` finds the transition probabilities by solving a single initial value problem for each unique covariate pattern. Specifically, an initial boundary problem of the form

$$\frac{d\mathbf{P}(t_0, t; \mathbf{z})}{dt} = \mathbf{P}(t_0, t; \mathbf{z})\mathbf{Q}(t; \mathbf{z}) \text{ for } \mathbf{P}(t_0, t_0; \mathbf{z}) = \mathbf{I} \tag{4.1}$$

is solved, for all times in the set of start and end times for intervals with covariate pattern $\mathbf{z}$ and where t_0 is the first start time in the set. By the identity $\mathbf{P}(t_0, t_2; \mathbf{z}) = \mathbf{P}(t_0, t_1; \mathbf{z})\mathbf{P}(t_1, t_2; \mathbf{z})$ any individual transition probability can be obtained from the solutions to (4.1)

$$\mathbf{P}(t_1, t_2; \mathbf{z}) = \mathbf{P}(t_0, t_1; \mathbf{z})^{-1}\mathbf{P}(t_0, t_2; \mathbf{z}).$$

However, this method relies on $\mathbf{P}(t_0, t_1; \mathbf{z})$ being invertible. The transition probability matrix may become singular if t_0 and t_1 are far apart and, for instance, the probability of remaining in a state over that time gets very close to zero. A singular matrix is most likely when poor starting values have been supplied to the optimization, but could also happen for data sets where the vast majority of individuals progress during the follow-up time.

If the algorithm fails due to a singular matrix, an error message will appear informing between which times (t_0, t_1) the singularity arose. Singularities can be avoided by using the `splits` option within `nhm.control`. Specifically, by supplying a vector of split times, `nhm` will categorize the start times into groups and solve separate initial value problems for any intervals within start times in the different groups. For instance, if a split is included at $t = 5$ then $P(6, 11)$ will be found by solving an initial value problem starting at the first time beyond 5 rather than from the smallest time with that covariate pattern. Adding splits will tend to increase the computation time, but not substantially.

The control option `safe` forces the functions to always solve a new initial value problem for each unique combination of start time t_0 and covariate pattern. While this avoids the need to define `splits`, it will typically lead to a significant increase in computation time. Note that depending on the model and starting values, the algorithm may still fail if appropriate parameter constraints have not been set (see Section 4.2.2).

4.2.4 Coarsening covariates

Usually, the majority of the computation time in evaluating the likelihood arises through the solution of the differential equations. The necessity to solve a separate system of equations for each covariate pattern means that models with continuous covariates can be substantially more computationally demanding.

It may be desirable for large datasets with continuous covariates to consider an approximation to the likelihood based upon coarsening the set of unique covariate values. Titman (2011) [13] proposed a simple method based on using K-means clustering to group similar values of the continuous covariates and assuming approximating transition probabilities within the same cluster by the transition probability that would arise from the mean covariate values

within that cluster. While coarsening the covariates will tend to introduce some attenuation bias in the covariate effects, it may be useful either in the model building stage or in cases where the full model is computationally impractical.

4.2.5 Parallelization

An alternative way to speed up the computation of mdoels for data with a large number of unique covariate patterns is to exploit parallel processing. The `ncores` option allows the `mclapply` function in the **parallel** package to allow multiple systems of ODEs to be solved simultaneously.

For models with misclassification, `mclapply` is also used to simultaneously perform the forward algorithm on multiple separate individuals.

By default, if `ncores`> 1 and `obsinfo=TRUE` then the package will parallelize across the complete gradient calls used to compute the observed Fisher information via finite differences. This should generally be more efficient as there should be less waiting time for all cores to complete. However, for large datasets this could cause memory issues. If `parallel_hess=FALSE`, the package will instead continue to exploit parallel processing within each gradient call in the same way as during the main optimization.

4.3 Fixed parameters

While `model.nhm` allows parameters to be fixed to zero (e.g. no transition, no time non-homogeneity, no covariate effect etc.) or for parameters to be constrained to be equal (e.g. the same time effect for $1 \rightarrow 2$ as $2 \rightarrow 3$), there may also be situations where we wish to fix parameters to specific values. This can be achieved using the `fixedpar` option in `nhm`. `fixedpar` should be vector of integers identifying the indices of the parameters to be fixed. In the optimization these parameters will be fixed at the values supplied in `initial`. Note that currently if `fixedpar` is used then several of the model output functions are unavailable (`predict`, `qmatrix.nhm`, `ematrix.nhm` and `initialprob.nhm`).

4.3.1 Example: Gompertz model

A time homogeneous model for the data introduced in Section 3.6 can be fitted using **msm**.

```
model_msm <- msm(state~time,gen.inits=TRUE,subject=id,data=example_data1,qmatrix=trans)
```

This can be used to gain estimates of the baseline parameters.

```
model_msm$estimates
```

```
     qbase      qbase      qbase
-0.5764525 -0.4078982 -0.6565766
```

```
initpar <- c(model_msm$estimates,rep(0,5))
gomp_fit1 <- nhm(gomp_model,initial=initpar)
```

Since no control options are specified, a BHHH optimization algorithm is performed which converges in 7 iterations. However, a further 16 gradient evaluations are required to compute the observed information, taking the majority of the computation time.

Alternatively, a Fisher scoring algorithm may be used

```
initpar <- c(model_msm$estimates,rep(0,5))
gomp_fit1a <- nhm(gomp_model,initial=initpar,control=nhm.control(fishscore=TRUE)
```

From the same starting values, this converges in just 5 iterations. Again, a further 16 gradient evaluations are required for the observed information.

4.4 Score test option

De Stavola (1988) [3] proposed local score tests to assess the homogeneity assumption in Markov multi-state models. The advantage of the score test is that it is not necessary to fit a more complicated model in order to assess whether it may fit better than the basic model. In the original paper by De Stavola, a linear model of the form

$$q_{rs}(t;\epsilon) = q_{rs0} + \epsilon t$$

is assumed, with this formulation giving a closed form for the test statistic. However, through solution of the systems of differential equations, any non-homogeneous model can be considered. In general, assume the parameter vector can be partitioned as $\boldsymbol{\theta} = (\boldsymbol{\psi}, \boldsymbol{\beta})$ where $\boldsymbol{\beta}$ controls the time non-homogeneity.

The `score_test` option in `nhm` enables a score test to be performed of the form

$$H_0 : \boldsymbol{\beta} = \boldsymbol{\beta}_0 \text{ vs. } H_1 : \boldsymbol{\beta} \neq \boldsymbol{\beta}_0. \tag{4.2}$$

Typically $\boldsymbol{\beta}_0 = \mathbf{0}$, corresponding to a time homogeneous model.

If `score_test=TRUE` then the `nhm` function simply computes the gradient and Fisher information at the supplied initial values and stores them in an object of class `nhm_score`. The `print` method for this object is a function which computes and prints the output from a score test.

Suppose $\hat{\boldsymbol{\psi}}_0$ is the maximum likelihood estimate for the restricted model in which $\boldsymbol{\beta} = \boldsymbol{\beta}_0$, and denote $\hat{\boldsymbol{\theta}}_0 = (\hat{\boldsymbol{\psi}}_0, \boldsymbol{\beta}_0)$. The score statistic is then of the form

$$S = \mathbf{U}_{\boldsymbol{\beta}}^{'} \mathbf{I}^{\boldsymbol{\beta}\boldsymbol{\beta}} \mathbf{U}_{\beta}$$

where $U_\beta = \frac{\partial \log L(\hat{\boldsymbol{\theta}}_0)}{\partial \beta}$ and $\mathbf{I}^{\boldsymbol{\beta}\boldsymbol{\beta}}$ is the square matrix created from the entries of the inverse Fisher information corresponding to $\boldsymbol{\beta}$. Under the null $S \sim \chi^2_{|\boldsymbol{\beta}|}$. If `fishscore=TRUE` was specified in the control options of `nhm` then the expected Fisher information will be used in the test, while otherwise the squared derivatives estimate will be used.

By default, the function uses the stored `parclass` of the parameters and assumes that the score test is with respect to all parameters that are of `Nonhom` class. However, the user can specify which parameters are to be tested by using the `which_comp` option. This should identify the indices of the parameter vector that are to be tested.

In addition to computing the overall statistic S, the function also calculates individual z-statistics for each individual parameter where $z_i = \mathbf{U}_{\boldsymbol{\beta}_i}/\sqrt{I^{\boldsymbol{\beta}_i\boldsymbol{\beta}_i}}$ and each z_i has an asymptotic $N(0,1)$ distribution under the null.

Note that for the score test to give sensible results the supplied initial parameter vector needs to represent the MLE for the restricted model. The function will attempt to check whether this is the case by assessing the size of $\mathbf{U}_{\boldsymbol{\psi}}^{'}\mathbf{I}^{\boldsymbol{\psi}\boldsymbol{\psi}}\mathbf{U}_{\psi}$ and produces a warning if it is not close to 0.

The score test option is most useful as an exploratory step to determine which, if any transition intensities exhibit time non-homogeneity, and in which direction. However, it could also be used in other situations; for instance to assess whether covariate effects may be beneficial.

4.4.1 Example

Using the example data set, we can fit a time homogeneous model. This could be done substantially faster using *msm*, but here we will use *nhm* directly using the previous model settings, except modifying the `nonh` term

```
nonh0 <- matrix(0,4,4)
gomp_model0 <- model.nhm(state~time, data=example_data1, subject = id, covariates=c("cov1",
"cov2"),type="gompertz",trans=trans,nonh=nonh0,covm=covm)
gomp_model0

nhm model object for a Markov model.

Model for intensities has 5 unknown parameters and is of gompertz type.

                    Name  Type
1             Base: 1->2 Trans
2             Base: 2->3 Trans
3             Base: 3->4 Trans
4 Covariate: cov1 1 -> 2   Cov
5 Covariate: cov2 1 -> 2   Cov

gomp_fit0 <- nhm(gomp_model0,gen_inits=TRUE,control=nhm.control(obsinfo=FALSE))
```

Here `obsinfo=FALSE` because we only need the parameter estimates. The parameter estimates from this model then make up starting values for the model defined in `gomp_model1`, with the time trend parameters set to zero:

```
initial_hom <- c(gomp_fit0$par[1:3],rep(0,3),gomp_fit0$par[4:5])
```

We may then use `nhm` to perform a score test of whether $\beta_{12} = \beta_{23} = \beta_{34} = 0$:

```
gomp_score <- nhm(gomp_model, initial=initial_hom, score_test=TRUE, control=
nhm.control(fishscore=TRUE))
gomp_score
```

```
Score test for non-homogeneity components
               Stat  p-val
Trend: 1->2  3.2202 0.0013
Trend: 2->3 -0.0008 0.9994
Trend: 3->4 -0.2383 0.8117
Overall     10.4680 0.0150
```

This indicates that there is a strong effect of time on the $1 \rightarrow 2$ transition, but no evidence of an effect for the other two transitions.

5 Model outputs

5.1 Print method for *nhm* class objects

The `print` method for a fitted model of `nhm` class produces a table of parameter estimates with corresponding 95% confidence intervals. Note these are the raw parameter estimates ($\theta_{rs}, \beta_{rs}, \boldsymbol{\gamma}_{rs}, a_{rs}, \eta_{rs}, \boldsymbol{\tau}_{rs}$ in the notation given in Section 3).

If the model is not of `bespoke` type then the parameters will be automatically labelled based on their type and which transition intensities, misclassification probabilities or initial probabilities they govern. For `bespoke` type models, by default the labels will simply be of the form 'Bespoke Q parameter k'. If desired, more informative names can be produced by including a `parnames` attribute to the `intens` function. This needs to be a character vector of length equal to the number of parameters governing the model for the intensities.

5.2 Transition probability estimates: *predict*

The `predict` method for an `nhm` class (fitted model) object allows the transition probabilities from a particular start time, `time0`, and starting state, `state0`, and for a particular covariate value, `covvalue`. If omitted, `time0=0`, `state0=0` and `covvalue` will be taken as the covariate means. Note that the function can only compute the probabilities for one covariate pattern. Hence `covvalue` must be a vector of length corresponding to the number of covariates in the model and in the order given in the `covariates` argument of the `nhm.model` call.

By default the function will compute the transition probabilities at a vector of times between `time0` and the maximum follow-up time in the data of length 100. However, the `times` option may be used to supply the set of times.

By default `predict` will supply approximate pointwise 95% confidence intervals for each transition probability by using a logit transformation to provide intervals that lie within $[0, 1]$.

To change the nominal coverage of the limits, `coverage` can be changed from its default of 0.95. Potentially more accurate asymptotic confidence intervals can be obtained by setting `sim=TRUE`, which will mean the simulation

delta method [10] will be used to compute the limits, rather than the delta method. Note that this option is substantially slower than using the delta method because the system of differential equations will be solved `B` times, where the default is `B=1000`. If no confidence intervals are required, `ci=FALSE` may be specified.

5.3 Transition intensity estimates: *qmatrix.nhm*

If estimates of the transition intensities are required, rather than the probabilities, then the function `qmatrix.nhm` may be used with an object of `nhm` class. The function has very similar syntax to `predict` except that `time0` refers simply to the minimum time at which the transition intensities are to be computed.

By default the asymptotic confidence intervals are computed via the delta method using a $\log$ transformation to ensure that the intervals lie on $[0, \infty)$. Simulation delta method intervals can be obtained by setting `sim=TRUE`.

5.4 State occupation probabilities

The `state_occupation_probabilities.nhm` function computes the state occupation probabilities for a fitted model. Whereas the transition probabilities are based on starting at a specific state at the initial time, the state occupation probabilities are calculated based on an initial state probability distribution. For misclassification models, by default this will be the `initp` estimated or specified by the model. The state occupation probabilities are most useful for models involving left truncation (see Section 7).

5.5 *ematrix.nhm* and *initialprob.nhm*

For misclassification type hidden Markov models, particularly those with covariates, there may be interest in obtaining estimates of the matrix of misclassification probabilities for a given covariate pattern. This is the purpose of the `ematrix.msm` function. In addition to the name of the `nhm` object, the covariate pattern `covvalue` can be supplied. The function outputs the misclassification probability estimates and their standard errors.

Similarly, if there is uncertainty in the initial state vector (see Section 3.7.2) then `initialprob.nhm` can be used to obtain the estimates of the initial state probabilities for a given `covvalue`, and also gives the corresponding standard errors.

5.6 Plot method for *nhm* class objects

Plotting an `nhm` class object will call either `predict.nhm` or `qmatrix.nhm` in order to produce plots of either the transition probabilities (when `what='probabilities'` which is the default), transition intensities (when `what='intensities'`) or state occupation probabilities (when `what='stateoccup'`). In each case a multi-panel plot is produced with a plot corresponding to either each state, in the case of transition or state occupation probabilities, or each viable transition. The same control parameters used in `predict` or `qmatrix.nhm` can also

be specified in `plot`, such as the evaluation times, the covariate vector and (for transition probabilities) the initial state. By default pointwise 95% confidence intervals are produced via the delta method, but simulation delta method intervals may also be produced by specifying `sim=TRUE`.

5.7 Example: B-spline model with misclassification and covariate effects

To illustrate the different types of model output, we consider fitting a B-spline type model to the second example dataset. To limit the number of unknown parameters, we only allow non-homogeneity with respect to the $1 \to 2$ transition intensity.

```
trans <- rbind(c(0,1,0,0),c(0,0,2,0),c(0,0,0,3),rep(0,4))
nonh <- rbind(c(0,1,0,0),rep(0,4),rep(0,4),rep(0,4))
```

As in Section **??**, it is assumed that state misclassification can only occur between adjacent transient states

```
emat <- rbind(c(0,1,0,0),c(2,0,3,0),c(0,4,0,0),rep(0,4))
```

A general model is assumed for the covariate effects, allowing a separate proportional effect on each of the transition intensities

```
covm <- list(cov1 = rbind(c(0,1,0,0),c(0,0,2,0),c(0,0,0,3),rep(0,4)),
cov2 = rbind(c(0,4,0,0),c(0,0,5,0),c(0,0,0,6),rep(0,4)))
```

Since the model is to be of `bspline` type, it is necessary to also specify the location of the knot points. These are chosen to ensure that a reasonably similar number of $1 \to 2$ transitions occur between each pair of knots. The upper limit is taken below the maximum follow-up time.

```
splinelist <- list(c(0,2,10))
bspline_model <- model.nhm(state~time, data=example_data2, subject = id,type="bspline",
covariates=c("cov1","cov2"),trans=trans,nonh=nonh,emat=emat,covm=covm,splinelist=splinelist)
```

```
nhm model object for a misclassification hidden Markov model.

Model has 17 unknown parameters.
Model for intensities has 13 unknown parameters and is of bspline type.

                     Name   Type
1              Base: 1->2  Trans
2              Base: 2->3  Trans
3              Base: 3->4  Trans
4   Spline pars: 1->2...1 Nonhom
5   Spline pars: 1->2...2 Nonhom
6   Spline pars: 1->2...3 Nonhom
7   Spline pars: 1->2...4 Nonhom
8  Covariate: cov1 1 -> 2    Cov
9  Covariate: cov1 2 -> 3    Cov
10 Covariate: cov1 3 -> 4    Cov
11 Covariate: cov2 1 -> 2    Cov
12 Covariate: cov2 2 -> 3    Cov
13 Covariate: cov2 3 -> 4    Cov
14             Emat: 1->2   Emat
15             Emat: 2->1   Emat
16             Emat: 2->3   Emat
17             Emat: 3->2   Emat
```

To find reasonable starting values, we fit a time homogeneous model without covariates

```
null_mod <- model.nhm(state~time, data=example_data2, subject = id,type="gompertz",
  trans=trans,nonh=array(0,c(4,4)),emat=emat)
null_fit <- nhm(null_mod,initial=c(-2,-2.5,-3,-3,-3,-3,-3),
  control=nhm.control(obsinfo=FALSE))

initial_sp <- c(null_fit$par[1:3],rep(0,10),null_fit$par[4:7])
bspline_fit <- nhm(bspline_model,initial=initial_sp)
```

The BHHH algorithm converges in 8 iterations. The bulk of the computation time is used to compute the numerical Hessian which requires 34 further gradient evaluations (and can be skipped by setting `obsinfo=FALSE`.

By default plotting the fitted object will produce a four panel plot of the state occupation probabilities for the mean covariate values:

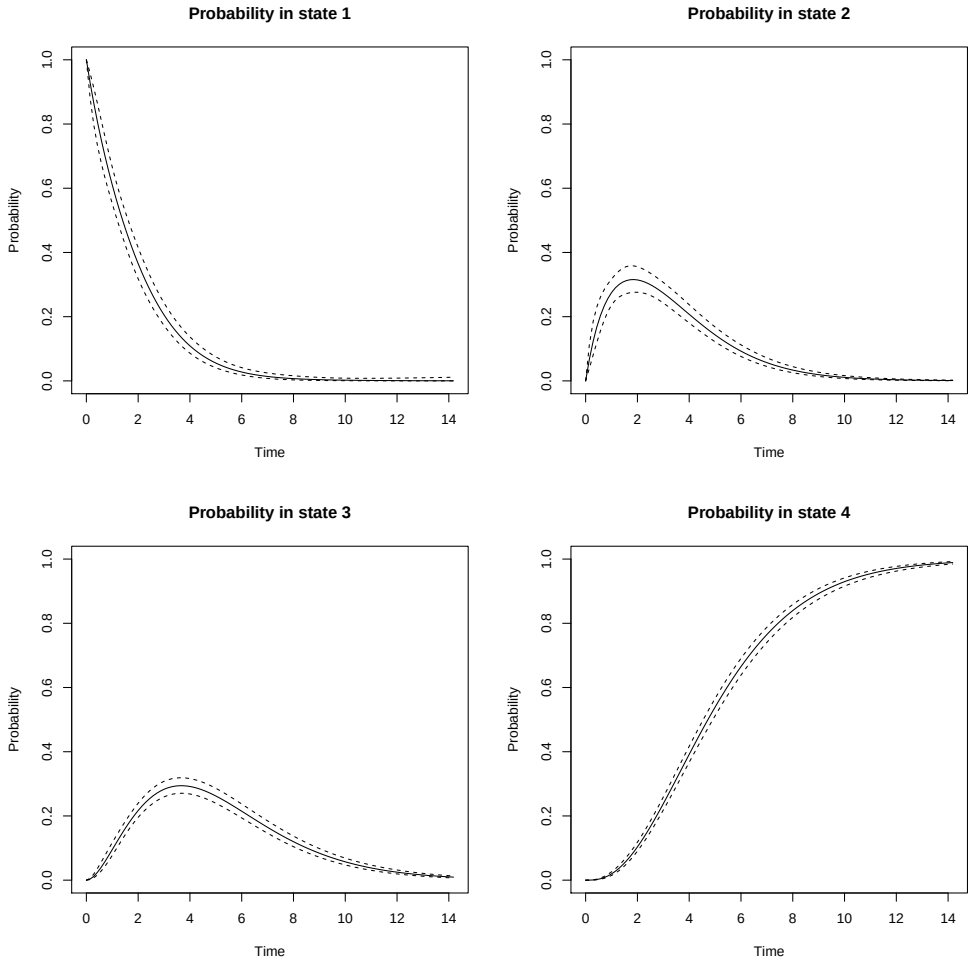
```
plot(bspline_fit)
```

However we can, for instance, produce a plot of the estimated transition intensities, for a subject with covariate values $(1, 0)$ by using:

```
plot(bspline_fit, what="intensities", covvalue=c(1,0))
```

There is apparently no obvious trend in the transition intensities, beyond some indication of an increased rate between 2 and 6 years.

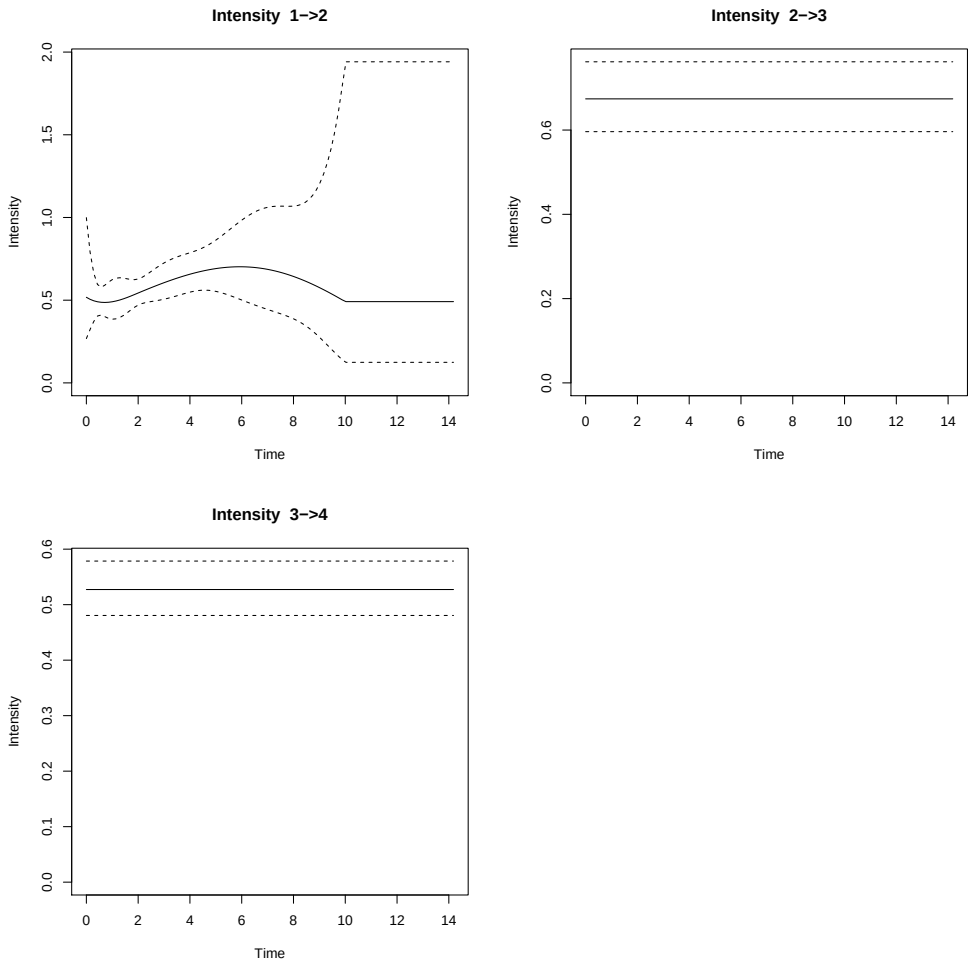
Similarly, the estimated matrix of misclassification probabilities can be obtained using




```
ematrix.nhm(bspline_fit)

$emat
           [,1]       [,2]       [,3] [,4]
[1,] 0.91896279 0.08103721 0.00000000    0
[2,] 0.07046095 0.89906616 0.03047288    0
[3,] 0.00000000 0.09200885 0.90799115    0
[4,] 0.00000000 0.00000000 0.00000000    1

$SEemat
           [,1]       [,2]       [,3] [,4]
[1,] 0.02495268 0.02495268 0.00000000    0
[2,] 0.02408139 0.02511198 0.01740597    0
[3,] 0.00000000 0.01904015 0.01904015    0
[4,] 0.00000000 0.00000000 0.00000000    0
```



6 Specifying bespoke models

The `bespoke` model type allows for user supplied functions for the generator matrix of intensities and its derivatives to be used with the package. There are two main motivations for supplying a bespoke function. Firstly, it allows models not otherwise accommodated within the package to be fitted. These could include intensities from other parametric families than Gompertz or Weibull, or models with interactions between covariates and time.

The second motivation is to allow the computation time to be reduced for models that are accommodated within the existing functionality. The automatically created functions are slower than a specifically written function, particular for models which are sparse in the number of possible time or covariate effects.

6.1 Example: Gompertz model

Suppose firstly we wish to fit the same model as in [...]. The function needs to take `t`, `z` and `x` as arguments, corresponding to the evaluation time, covariate vector and parameter vector, and should return a list elements `q` - the

intensities matrix and `qp` - the first derivatives of the intensities matrix.

```
fourstate_expgrowth<-function(t,z,x) {
  q12<-exp(x[1])
  q23<-exp(x[2])
  q34<-exp(x[3])
  i12<-q12*exp(x[4]*t + z[1]*x[7] + z[2]*x[8])
  i23<-q23*exp(x[5]*t)
  i34<-q34*exp(x[6]*t)
  q<-rbind(c(0,i12,0,0),c(0,0,i23,0),c(0,0,0,i34),c(0,0,0,0))
  diag(q) <- c(-i12,-i23,-i34,0)
  der<-array(0,c(4,4,8))
  der[1,1,1]<--i12
  der[1,2,1]<-i12
  der[2,2,2]<--i23
  der[2,3,2]<-i23
  der[3,3,3]<--i34
  der[3,4,3]<-i34
  der[,,4:6]<-t*der[,,1:3]
  der[1,1,7]<--i12*z[1]
  der[1,2,7]<-i12*z[1]
  der[1,1,8]<--i12*z[2]
  der[1,2,8]<-i12*z[2]
  Q<-list(q=q,qp=der)
  return(Q)
}
attr(fourstate_expgrowth,"npar")<-8
attr(fourstate_expgrowth,"parnames")<-c("1->2 base:","2->3 base:","3->4 base:","1->2 NH",
"2->3 NH","3->4 NH","Cov1","Cov2")
attr(fourstate_expgrowth,"parclass")<-c("Trans","Trans","Trans","Nonhom",
"Nonhom","Nonhom","Cov","Cov")
```

The attributes are not essential, but will improve the informativeness of the output.

```
gomp_model_bespoke <- model.nhm(state~time, data=example_data1, subject = id, covariates=
  c("cov1","cov2"),   type="bespoke",trans=trans,intens=fourstate_expgrowth)
initpar <- c(model_msm$estimates,rep(0,5))
gomp_fit1b <- nhm(gomp_model_bespoke,initial=initpar)
```

Exactly the same model as in `gomp_fit1` is fitted, but the computation time in this case is around ..% faster.

6.2 Example: Gompertz model with interaction covariate effects

Now we consider an extended version of the model, where the degree of time non-homogeneity depends on covariate

1. An extra parameter is added that introduces a common interaction between time and covariate 1 for the $1 \rightarrow 2$, $2 \rightarrow 3$ and $3 \rightarrow 4$ transitions.

```
fourstate_expgrowth_int<-function(t,z,x) {
  q12<-exp(x[1])
  q23<-exp(x[2])
  q34<-exp(x[3])
  i12<-q12*exp((x[4]+z[1]*x[9])*t + z[1]*x[7] + z[2]*x[8])
  i23<-q23*exp((x[5]+z[1]*x[9])*t)
  i34<-q34*exp((x[6]+z[1]*x[9])*t)
  q<-rbind(c(0,i12,0,0),c(0,0,i23,0),c(0,0,0,i34),c(0,0,0,0))
  diag(q) <- c(-i12,-i23,-i34,0)
  der<-array(0,c(4,4,9))
  der[1,1,1]<--i12
  der[1,2,1]<-i12
  der[2,2,2]<--i23
  der[2,3,2]<-i23
  der[3,3,3]<--i34
  der[3,4,3]<-i34
  der[,,4:6]<-t*der[,,1:3]
  der[1,1,7]<--i12*z[1]
  der[1,2,7]<-i12*z[1]
  der[1,1,8]<--i12*z[2]
  der[1,2,8]<-i12*z[2]
  der[1,1,9]<--i12*z[1]*t
  der[1,2,9]<-i12*z[1]*t
  der[2,2,9]<--i23*z[1]*t
  der[2,3,9]<-i23*z[1]*t
  der[3,3,9]<--i34*z[1]*t
  der[3,4,9]<-i34*z[1]*t
  Q<-list(q=q,qp=der)
  return(Q)
}
attr(fourstate_expgrowth_int,"npar")<-9
attr(fourstate_expgrowth_int,"parnames")<-c("1->2 base:","2->3 base:","3->4 base:","1->2 NH",
"2->3 NH","3->4 NH","Cov1","Cov2","TimeEfCov1")
attr(fourstate_expgrowth_int,"parclass")<-c("Trans","Trans","Trans","Nonhom",
"Nonhom","Nonhom","Cov","Cov","Cov")
```

Since this model is an extension of the previous one, it makes sense to use the estimates from the simpler model to obtain starting parameter estimates.

```
gomp_model_bespoke2 <- model.nhm(state~time, data=example_data1, subject = id, covariates=
  c("cov1","cov2"), type="bespoke",trans=trans,intens=fourstate_expgrowth_int)
initpar2 <- c(gomp_fit1b$par,0)
gomp_fit2 <- nhm(gomp_model_bespoke2,initial=initpar2)
gomp_fit2
```

```
               Est Low 95%  Up 95%
1->2 base: -0.9004 -1.0412 -0.7596
2->3 base: -0.3902 -0.5361 -0.2443
3->4 base: -0.6325 -0.8086 -0.4565
1->2 NH     0.0769  0.0294  0.1244
2->3 NH    -0.0086 -0.0473  0.0300
3->4 NH    -0.0096 -0.0459  0.0267
Cov1        0.4760  0.3293  0.6228
Cov2       -0.3179 -0.3899 -0.2458
TimeEfCov1  0.0077 -0.0149  0.0304

Deviance: 5789.49
```

In this case, the additional covariate does not improve the fit, which may be verified using `anova`

```
anova(gomp_fit1b,gomp_fit2)
```

```
$LRS
[1] 0.4440297

$df
[1] 1

$p
[1] 0.5051839
```

7 Left truncation

For multi-state data arising from cohort studies, the natural choice of time scale may be patient age, rather than time in the study. Moreover, the aim of the analysis may be to obtain an estimate of the target population's transition intensities as a function of age. However, patients will only be recruited into the study conditional on survival to their age at the calendar date the study began which we can denote as t_{0i}. Depending on the inclusion criteria or recruitment method, entry into the study may also be conditional on occupation of a specific state (or subset of the non-absorbing states). In general we can denote this set of states as $\mathcal{L}$ which should be a subset of $\{1, \ldots, R\}$.

For models without misclassification, if the state at a patient's entry age is known (either because it is fixed, e.g. state 1, or because their status is observed at entry) then it is not necessary to do anything except include (or, where necessary, impute) their state at entry in the supplied data.

In contrast, if the state at t_{0i} is not known or is subject to misclassification error, the left truncation needs to be accounted for in the likelihood. Specifically, we need to define a baseline age, t_{00}, where there is a fixed state occupation distribution (this could be all in state 1 with probability 1 or it can be defined via the initial state distribution options given in Section 3.7). The state occupation probability at t_{0i} for a state $j \in \mathcal{L}$ is then given by

$$\pi_{ij}(t_{0i}) = P(X_i(t_{0i}) = j) = \frac{\sum_k \pi_{ik}^{(0)} P_{kj}(t_{00}, t_{0i})}{\sum_{l \in \mathcal{L}} \sum_k \pi_{ik}^{(0)} P_{kl}(t_{00}, t_{0i})}$$

while $\pi_{ij}(t_{0i}) = 0$ for $j \notin \mathcal{L}$. Here, we make the assumption that patients are recruited as a representative sample of those within state $\mathcal{L}$ at time t_{0i}.

By default, t_{0i} is ascertained from the patient time given in the data for the patient's first observation. However, some datasets may only include patients who had at least m visits/observations.

Since version 0.1.2, this form of left-truncation can be accommodated within the modelling provided the overall model is of misclassification type. As noted above, a model can be forced to be treated as misclassification type by supplying a emat argument that is an $R \times R$ matrix of zeroes. `firstobs='misc'` must be specified, but this should not be a restriction since censoring codes can be used for other situations.

Left truncation options

When specifying the model using `model.nhm`, the relevant arguments are

- `ltruncation_states` which should be a vector corresponding to $\mathcal{L}$ (the set of states which a patient could occupy at time of recruitment into the model).
- `ltruncation_time` which should be a scalar that specifies the baseline age t_{00} and defaults to 0.
- `ltruncation_entry` which should be a positive integer indicating at which observation number the left truncation conditions should apply. By default this is 1, if it is m then it is assumed that the patient has been recruited into the study conditional on $X(t_{0i}) \in \mathcal{L}$ where t_{0i} is their age at their mth observation.

If the state occupation probabilities are calculated using `state_occupation_probabilities.nhm` (or `plot` using `what='stateoccup'`) for a model with left truncation then the initial state occupation probabilities at time `time0` will be calculated assuming the initial state probabilities held at `ltruncation_time` and that only occupation within `ltruncation_states` is possible at `time0`. Note that it is also possible to supply a list `ltruncation` containing `ltruncation_states` and `ltruncation_time` in order to estimate the state occupation probabilities that would arise under different sampling schemes.

7.1 Additional options

Version 0.1.2. includes additional developmental options to

1. allow testing of the assumption of non-informative observation via targeted score tests (using the `inform` option in `model.nhm`)
2. allow semi-Markov and misclassification hidden semi-Markov models defined as aggregated Markov processes (using the `phasemap` option). Currently this can only be applied to 'bespoke' type models.
3. estimate the expected time spent in each state of a fitted model, using the `state_life_expectancy` function, and to estimate the expected hitting time for a fitted progressive model using `expected_hitting_time`.

Further details of these features will be provided in the future.

References

[1] Berndt E, Hall B, Hall R, Hausman J. (1974). Estimation and Inference in Nonlinear Structural Models. *Annals of Economic and Social Measurement.* **3**. 653–665.

[2] Cox DR, Miller HD (1965).*The Theory of Stochastic Processes.* Chapman and Hall, London.

[3] de Stavola BL. (1988). Testing Departures from Time Homogeneity in Multistate Markov Processes. *Journal of the Royal Statistical Society: Series C (Applied Statistics)* **37**. 242–250.

[4] Gay D.M. (1990), Usage summary for selected optimization routines. Computing Science Technical Report 153, AT&T Bell Laboratories, Murray Hill.

[5] Henningsen A, Toomet O. (2011). maxLik: A package for maximum likelihood estimation in R. *Computational Statistics*, **26**. 443–458.

[6] Jackson C.H. (2011). Multi-State Models for Panel Data: The msm Package for R. *Journal of Statistical Software*, **38**(8), 1–29. `http://www.jstatsoft.org/v38/i08/`.

[7] Jackson C.H, Sharples L.D., Thompson S.G, Duffy S.W, Couto E. (2003) Multistate Markov models for disease progression with classification error. *Journal of the Royal Statistical Society Series D* **52**:193-209

[8] Kalbfleisch J.D, Lawless J.F. (1985) The analysis of panel data under a Markov assumption. *Journal of the American Statistical Association* **80**:863-871.

[9] Lystig TC., Hughes JP. (2002) Exact computation of the observed information matrix for hidden Markov models. *Journal of Computational and Graphical Statistics*, **11**, 678–689.

[10] Mandel M. (2013) Simulation-based confidence intervals for functions with complicated derivatives. *The American Statistician*, **67**. 76–81.

[11] Petzold, Linda R. (1983) Automatic Selection of Methods for Solving Stiff and Nonstiff Systems of Ordinary Differential Equations. *Siam Journal of Scientific and Statistical Computing*, **4**. 136–148.

[12] Soetaert K, Petzoldt T, Woodrow Setzer R. (2010) Solving differential equations in R: Package deSolve. *Journal of Statistical Software*, **33**. 1–25

[13] Titman A.C. (2011). Flexible Nonhomogeneous Markov Models for Panel Observed Data. *Biometrics*, **67**. 780–787.