1 Overview

In this work, we describe the basic usage of the hierarchical formulation of the ctsem software for continuous-time dynamic modelling in R (R Core Team, 2014). This formulation was described in detail in Driver and Voelkle (2018a), though since then the scope has been expanded to include regular (i.e. not Bayesian) maximum likelihood estimation, nonlinear models (such as parameters that vary over time), and optimization with optional importance sampling. The latter enhancement means that the approach described herein largely supersedes the initial mixed effects approach based upon OpenMx (Neale et al., 2016), as estimation options now include maximum likelihood, maximum a posteriori, and fully Bayesian with the Stan software's (Carpenter et al., 2017; Stan Development Team, n.d.) Hamiltonian Monte Carlo sampler. The major use case for the OpenMx based wide formulation is many subjects, few time points, and most or all time intervals consistent across subjects. This older structural equation formulation is discussed in detail in Driver et al. (2017). Although some of the R commands and functionality are now different in the newer Stan based format, the general concepts discussed in the original work are still relevant background. Additional material can be found at https://github.com/cdriveraus/ctsem (very quick start guide on main page), https://github.com/cdriveraus/ctsem/discussions (question and answer), and https://cdriver.netlify.app/ (tutorials and Bayesian quick start).

ctsem allows for specification and fitting of a range of continuous and discrete time models to study dynamics and processes within a subject, as well as individual differences in these dynamics. The models may include multiple indicators (dynamic factor analysis), multiple, interrelated, potentially higher order processes, and both time dependent and time independent covariates. Classic longitudinal models like latent growth curve and latent change score models can be formulated as special cases. This approach offers some advantages over what could be considered more typical dynamic modelling approaches, such as vector autoregressive models or latent change score models. Two important advantages relate to the handling of time, and the treatment of individual differences. Time information is explicitly incorporated into the model, such that predictions from one measurement to another relate exactly to the amount of time that has passed, rather than simply the number of measurements, as is typical in discrete-time models. When the time interval between all measurements are equal, there is an exact relationship between the discrete and continuous time form (in many cases), but when they differ, the continuous-time form is typically more appropriate. For more on continuous-time models, see Oud and Jansen (2000), Singer (1993) and Voelkle and Oud (2013). With respect to individual differences, the hierarchical Bayesian approach allows for individual variation across all model parameters, while making full use of the data from all subjects to enhance estimation. This has the result that reasonable individual specific parameter estimates may be obtained with far fewer time points than would be required by single-subject time-series type modelling approaches. For more on hierarchical Bayesian models, see Gelman et al. (2014).

This document is structured such that we briefly describe the continuous time dynamic model governing within subject dynamics, and the hierarchical model governing the distribution of subject level parameters. A more conceptual overview along with technical details is given in Driver and Voelkle (2018a). Following, we walk through installing the ctsem software, setting up a data structure, specifying and fitting the model, followed by summary and plotting functions. Some details on additional complexity are then provided, including an example model with a more complex dynamic structure, a discussion of the various options for incorporating stationarity assumptions into the model, and a walk-through of the various transformations involved in the model.

1.1 Subject level latent dynamic model

This section describes the subject level model characterising the system dynamics and measurement properties. Although we do not describe it explicitly, the corresponding discrete time autoregressive / moving average models can be specified and use the same set of parameter matrices we describe. The subject level dynamics are described by the following stochastic differential equation:

$$\mathrm{d}\boldsymbol{\eta}(t) = \Big(\mathbf{A}\boldsymbol{\eta}(t) + \mathbf{b} + \mathbf{M}\boldsymbol{\chi}(t)\Big)\mathrm{d}t + \mathbf{G}\mathrm{d}\mathbf{W}(t) \tag{1}$$

Vector $\boldsymbol{\eta}(t) \in \mathbb{R}^v$ represents the state of the latent processes at time t. The matrix $\mathbf{A} \in \mathbb{R}^{v\times v}$ (DRIFT) represents the drift matrix, with auto effects on the diagonal and cross effects on the off-diagonals characterizing the temporal dynamics of the processes.

The continuous time intercept vector $\mathbf{b} \in \mathbb{R}^v$ (CINT), in combination with $\mathbf{A}$, determines the long-term level at which the processes fluctuate around.

Time dependent predictors $\boldsymbol{\chi}(t)$ represent inputs to the system that vary over time and are independent of fluctuations in the system. Equation 1 shows a generalized form for time dependent predictors, that could be treated a variety of ways dependent on the assumed time course of time dependent predictors and their effects. We use a simple impulse form shown in Equation 2, in which the predictors are treated as impacting the processes only at the instant of an observation occasion u. When necessary, the evolution over time can be modeled by extending the state matrices, for examples and discussion see Driver and Voelkle (2018b).

$$\boldsymbol{\chi}(t) = \sum_{u\in\mathbf{U}} \mathbf{x}_u \delta(t - t_u) \tag{2}$$

Here, time dependent predictors $\mathbf{x}_u \in \mathbb{R}^l$ (tdpreds) are observed at measurement occasions $u \in \mathbf{U}$. The Dirac delta function $\delta(t - t_u)$ is a generalized function that is ∞ at 0 and 0 elsewhere, yet has an integral of 1 (when 0 is in the range of integration). It is useful to model an impulse to a system, and here is scaled by the vector of time dependent predictors $\mathbf{x}_u$. The effect of these impulses on processes $\boldsymbol{\eta}(t)$ is then $\mathbf{M} \in \mathbb{R}^{v\times l}$ (TDPREDEFFECT).

$\mathbf{W}(t) \in \mathbb{R}^v$ (DIFFUSION) represents independent Wiener processes, with a Wiener process being a random-walk in continuous time. $\mathrm{d}\mathbf{W}(t)$ is meaningful in the context of stochastic differential equations, and represents the stochastic error term, an infinitesimally small increment of the Wiener process. Lower triangular matrix $\mathbf{G} \in \mathbb{R}^{v\times v}$ represents the effect of this noise on the change in $\boldsymbol{\eta}(t)$. $\mathbf{Q}$, where $\mathbf{Q} = \mathbf{G}\mathbf{G}^\top$, represents the variance-covariance matrix of the diffusion process in continuous time.

1.2 Subject level measurement model

The latent process vector $\boldsymbol{\eta}(t)$ has measurement model:

$$\mathbf{y}(t) = \boldsymbol{\Lambda}\boldsymbol{\eta}(t) + \boldsymbol{\tau} + \boldsymbol{\epsilon}(t) \quad \text{where } \boldsymbol{\epsilon}(t) \sim \mathrm{N}(\mathbf{0}_c, \boldsymbol{\Theta}) \tag{3}$$

$\mathbf{y}(t) \in \mathbb{R}^c$ is the vector of manifest variables, $\boldsymbol{\Lambda} \in \mathbb{R}^{c\times v}$ (LAMBDA) represents the factor loadings, and $\boldsymbol{\tau} \in \mathbb{R}^c$ (MANIFESTMEANS) the manifest intercepts. The residual vector $\boldsymbol{\epsilon} \in \mathbb{R}^c$ has covariance matrix $\boldsymbol{\Theta} \in \mathbb{R}^{c\times c}$ (MANIFESTVAR).

1.3 Overview of hierarchical model

Parameters for each subject are first drawn from a simultaneously estimated higher level distribution over an unconstrained space, then a set of parameter specific transformations are applied so that a) each parameter conforms to necessary bounds and b) is subject to the desired prior. Following this, in some cases matrix transformations are applied to generate the continuous time matrices described. The higher level distribution has a multivariate normal prior (though priors may be switched off as desired). We provide a brief description here, and an R code example later in this work, but for the full details see Driver and Voelkle (2018a).

The joint-posterior distribution of the model parameters given the data is as follows:

$$p(\boldsymbol{\Phi}, \boldsymbol{\mu}, \mathbf{R}, \boldsymbol{\beta} | \mathbf{Y}, \mathbf{z}) \propto p(\mathbf{Y}|\boldsymbol{\Phi}) p(\boldsymbol{\Phi}|\boldsymbol{\mu}, \mathbf{R}, \boldsymbol{\beta}, \mathbf{z}) p(\boldsymbol{\mu}, \mathbf{R}, \boldsymbol{\beta}) \tag{4}$$

Subject specific parameters $\boldsymbol{\Phi}_i$ are determined in the following manner:

$$\boldsymbol{\Phi}_i = \text{tform}\Big(\boldsymbol{\mu} + \mathbf{R}\mathbf{h}_i + \boldsymbol{\beta}\mathbf{z}_i\Big) \tag{5}$$

$$\mathbf{h}_i \sim \text{N}(\mathbf{0}, \mathbf{1}) \tag{6}$$

$$\boldsymbol{\mu} \sim \text{N}(\mathbf{0}, \mathbf{1}) \tag{7}$$

$$\boldsymbol{\beta} \sim \text{N}(\mathbf{0}, \mathbf{1}) \tag{8}$$

$\boldsymbol{\Phi}_i \in \mathbb{R}^s$ is the s length vector of parameters for the dynamic and measurement models of subject i. $\boldsymbol{\mu} \in \mathbb{R}^s$ parameterizes the means of the raw population distributions of subject level parameters. $\mathbf{R} \in \mathbb{R}^{s \times s}$ is the matrix square root of the raw population distribution covariance matrix, parameterizing the effect of subject specific deviations $\mathbf{h}_i \in \mathbb{R}^s$ on $\boldsymbol{\Phi}_i$. $\boldsymbol{\beta} \in \mathbb{R}^{s \times w}$ is the raw effect of time independent predictors $\mathbf{z}_i \in \mathbb{R}^w$ on $\boldsymbol{\Phi}_i$, where w is the number of time independent predictors. $\mathbf{Y}_i$ contains all the data for subject i used in the subject level model – $\mathbf{y}$ (process related measurements) and $\mathbf{x}$ (time dependent predictors). $\mathbf{z}_i$ contains time independent predictors data for subject i. tform is an operator that applies a transform to each value of the vector it is applied to. The specific transform depends on which subject level parameter matrix the value belongs to, and the position in that matrix.

At a number of points, we will refer to the parameters prior to the tform function as 'raw' parameters. So for instance 'raw population standard deviation' would refer to a diagonal entry of $\mathbf{R}$, and 'raw individual parameters for subject i' would refer to $\boldsymbol{\mu} + \mathbf{R}\mathbf{h}_i + \boldsymbol{\beta}\mathbf{z}_i$. In contrast, without the 'raw' prefix, 'population means' would refer to tform($\boldsymbol{\mu}$), and would typically reflect values the user is more likely to be interested in, such as the continuous time drift parameters.

1.4 Install software and prepare data

Install ctsem software within R:

```
install.packages("ctsem")
library("ctsem")
```

Prepare data in long format, each row containing one time point of data for one subject. We need a subject id column, named by default "id", though this can be changed in the model specification.

Some of the outputs are simpler to interpret if subject id is a sequence of integers from 1 to the number of subjects, but this is not a requirement. We also need a time column "time", containing positive numeric values for time, columns for manifest variables (the names of which must be given in the next step using `ctModel`), columns for time dependent predictors (these vary over time but have no model estimated and are assumed to impact latent processes instantly), and columns for time independent predictors (which predict the subject level parameters, that are themselves time invariant – thus the values for a particular time independent predictor must be the same across all observations of a particular subject).

```
    id  time   Y1    Y2 TD1     TI1      TI2    TI3
6    1 0.594 7.50 12.34   0 -0.0538 -1.36630 -0.207
7    1 0.889 6.83 12.63   1 -0.0538 -1.36630 -0.207
8    1 1.187 7.01 11.91   0 -0.0538 -1.36630 -0.207
101  2 2.482 2.33  1.50   0      NA  1.52461 -0.325
111  2 2.720 3.26 -1.16   0      NA  1.52461 -0.325
121  2 2.904 3.13 -1.34   0      NA  1.52461 -0.325
131  2 3.152 1.42 -1.28   0      NA  1.52461 -0.325
42   3 0.000 7.01  9.67   0 -0.9261  0.00812  0.201
52   3 0.326 7.02 10.09   0 -0.9261  0.00812  0.201
```

As will be discussed in detail later, default priors for the model are set up with an attempt to be 'weakly informative' for typical applications in the social sciences, on data that is centered and scaled. Because of this, we recommend grand mean centering and scaling each variable in the data, with the exception of time dependent predictors, which should be scaled, but centered such that a value of zero implies no effect. This exception is because time dependent predictors are modelled as impulses to the system with no persistence through time, at all times when not observed the value is then inherently zero. Similarly, we expect a time interval of 1.00 to reflect some 'moderate change' in the underlying process. If we wished to model daily hormonal fluctuations, with a number of measurements each day, a time scale of hours, days, or weeks could be sensible – minutes or years would likely be problematic. If the data are not adjusted according to these considerations, the priors themselves should be adjusted, or at least their impact carefully considered – though note also that an inappropriate scaling of time can also result in numerical difficulties, regardless of priors.

```
ctstantestdat[,c('Y1','Y2','TI1','TI2','TI3')] <-
 scale(ctstantestdat[,c('Y1','Y2','TI1','TI2','TI3')])
```

Functions to convert between wide and long formats used by ctsem are available, these are `ctWideToLong`, `ctDeintervalise`, `ctLongToWide`, `ctIntervalise`. For details see the relevant help in R.

1.5 Missing values

Missingness in the manifest variables is handled using the typical filtering / full information maximum likelihood approach. Missing values on the covariates – either time dependent or independent – are somewhat more problematic, and will cause an error by default. Various alternative approaches are available: In certain cases it may be reasonable to replace missing values with zeros, or to sample them as part of the model. See the ctStanFit help for more details.

1.6 Model specification

Specify model using `ctModel(type="stanct",...)`. `"stanct"` specifies a continuous time model in Stan format, `"standt"` specifies discrete time, while `"omx"` is the original ctsem behaviour and prepares an OpenMx model Driver et al. (discussed in 2017). Other arguments to `ctModel` include the list of within-subject parameter matrices, with LAMBDA, the factor loading matrix, the only one that must be specified. All other matrices are set to the system dimensions implied by LAMBDA (number of columns representing latent processes, rows representing manifest indicators), and in general filled with free parameters. The CINT matrix is the only exception to this, it is filled with zero's by default for identification purposes. Such parameter matrices can be manually specified using either numeric values when an element of the model should be fixed, a character string when it should be estimated, or a character string containing at least one square bracket for deterministic relations between parameter matrices and or latent states. The following code demonstrates this by freeing the CINT matrix of latent intercepts, and instead fixing the MANIFESTMEANS matrix of manifest intercepts. See the help for `ctModel`, available in R via `?ctModel`, for more details.

```
model<-ctModel(type='stanct',
  latentNames=c('eta1','eta2'),
  manifestNames=c('Y1','Y2'),
  CINT=matrix(c('cint1','cint2 |||| TI1'),nrow=2,ncol=1),
  MANIFESTMEANS=matrix(c(0,0),nrow=2,ncol=1),
  TDpredNames='TD1',
  TIpredNames = 'TI1',tipredDefault = FALSE,
  LAMBDA=diag(2))
```

It is possible to simplify the specification of parameter matrices somewhat, by passing a vector of values and or parameters to be put row-wise (distinct from the R default of column-wise) into a matrix of the correct size. Singular values can also be used as a fill for the entire matrix. The above model in this shorter form is specified like:

```
model<-ctModel(type='stanct',
  latentNames=c('eta1','eta2'),
  manifestNames=c('Y1','Y2'),
  CINT=c('cint1','cint2 |||| TI1'),
  MANIFESTMEANS=0,
  TDpredNames='TD1',
  TIpredNames = 'TI1',tipredDefault = FALSE,
  LAMBDA=c(1,0,0,1))


     [,1] [,2]
[1,]  "1"  "0"
[2,]  "0"  "1"
     [,1]
[1,]  "0"
[2,]  "0"
     [,1]
[1,]          "cint1"
[2,] "cint2 |||| TI1"
```

The code above specifies a first order bivariate latent process model, with each process measured by a single, potentially noisy, manifest variable. A single time dependent predictor is included in

Table 1: ctModel arguments

Argument	**Sign**	**Default**	**Meaning**
LAMBDA	Λ		n.manifest $\times$ n.latent loading matrix relating latent to manifest variables.
n.manifest	c		Number of manifest indicators per individual at each measurement occasion.
n.latent	v		Number of latent processes.
manifestNames		Y1, Y2, etc	n.manifest length character vector of manifest names.
latentNames		eta1, eta2, etc	n.latent length character vector of latent names.
T0VAR	Q_1^*	free	lower tri n.latent $\times$ n.latent matrix of latent process initial covariance, specified with standard deviations on diagonal and covariance related parameters on lower triangle.
T0MEANS	η_1	free	n.latent $\times$ 1 matrix of latent process means at first time point, T0.
MANIFESTMEANS	τ	free	n.manifest $\times$ 1 matrix of manifest intercepts.
MANIFESTVAR	Θ	free	diagonal matrix of var / cov between manifests, specified with standard deviations on diagonal and zeroes elsewhere.
DRIFT	A	free	n.latent $\times$ n.latent matrix of continuous auto and cross effects.
CINT	b	0	n.latent $\times$ 1 matrix of continuous intercepts.
DIFFUSION	Q	free	lower triangular n.latent $\times$ n.latent matrix containing standard deviations of latent process on diagonal, and covariance related parameters on lower off-diagonals.
n.TDpred	l	0	Number of time dependent predictors in the dataset.
TDpredNames		TD1, TD2, etc	n.TDpred length character vector of time dependent predictor names.
TDPREDEFFECT	M	free	n.latent $\times$ n.TDpred matrix of effects from time dependent predictors to latent processes.
n.TIpred	p	0	Number of time independent predictors.
TIpredNames		TI1, TI2, etc	n.TIpred length character vector of time independent predictor names.
PARS		free	0×0 Matrix containing additional parameters to be used, for instance in nonlinear models.

the model, as well as a single time independent predictor with effects restricted to only the second continuous intercept. To see the between and within subject model equations, the `ctModelLatex` function can be used on the model object, generating output as shown in Figure 1.

Table 1 shows the basic arguments one may consider and their link to the dynamic model parameters. Note that ctModel requires variance covariance matrices (DIFFUSION, T0VAR, MANIFESTVAR) to be specified in a matrix square root form, with standard deviations on the diagonal, covariance related parameters on the lower off diagonal (excluding MANIFESTVAR which should for many casess be diagonal only), and (typically) zeroes on the upper off diagonal. While it is possible to fix the lower off diagonals of covariance related matrices to non-zero values, in general this is difficult to interpret because of the necessary matrix transformations, thus we recommend either to fix to zero, or leave free.

The `pars` subobject of the created model object (in this case, `model$pars`) shows the parameter specification, including both fixed and free parameters, whether the parameters vary across individuals, how the parameter is transformed from a standard normal distribution (thus setting both priors and bounds), and whether that parameter is regressed on the time independent predictors.

Subject parameter distribution:

$$\underbrace{\begin{bmatrix}\text{cint1}_i\\ \text{cint2}_i\end{bmatrix}}_{\phi(i)} \sim \text{N}\left(\begin{bmatrix}\text{raw\_cint1}\\ \text{raw\_cint2}\end{bmatrix}, \begin{bmatrix}\text{rawPCov\_3\_1} & \text{rawPCov\_3\_2} & \text{rawPCov\_3\_3} & \text{rawPCov\_4\_3}\\ \text{rawPCov\_4\_1} & \text{rawPCov\_4\_2} & \text{rawPCov\_4\_3} & \text{rawPCov\_4\_4}\end{bmatrix}\right)$$

Initial latent state:

$$\underbrace{\begin{bmatrix}\text{eta1}\\ \text{eta2}\end{bmatrix}(t_0)}_{\boldsymbol{\eta}(t_0)} \sim \text{N}\left(\underbrace{\begin{bmatrix}\text{T0m\_eta1}\\ \text{T0m\_eta2}\end{bmatrix}}_{\text{T0MEANS}}, \underbrace{UcorSDtoCov\left\{\begin{bmatrix}0.001 & 0\\ 0 & 0.001\end{bmatrix}\right\}}_{\underset{\text{T0VAR}}{\mathbf{Q}^*_{t0}}}\right)$$

Deterministic change:

$$\underbrace{\mathrm{d}\begin{bmatrix}\text{eta1}\\ \text{eta2}\end{bmatrix}(t)}_{\mathrm{d}\boldsymbol{\eta}(t)} = \left(\underbrace{\begin{bmatrix}\text{drift\_eta1} & \text{drift\_eta1\_eta2}\\ \text{drift\_eta2\_eta1} & \text{drift\_eta2}\end{bmatrix}}_{\underset{\text{DRIFT}}{\mathbf{A}}}\underbrace{\begin{bmatrix}\text{eta1}\\ \text{eta2}\end{bmatrix}(t)}_{\boldsymbol{\eta}(t)} + \underbrace{\begin{bmatrix}\text{cint1}\\ \text{cint2}\end{bmatrix}}_{\underset{\text{CINT}}{\mathbf{b}}} + \underbrace{\begin{bmatrix}\text{td\_eta1\_TD1}\\ \text{td\_eta2\_TD1}\end{bmatrix}}_{\underset{\text{TDPREDEFFECT}}{\mathbf{M}}}\underbrace{\begin{bmatrix}\text{TD1}\end{bmatrix}}_{\boldsymbol{\chi}(t)}\right)\mathrm{d}t \quad +$$

Random change:

$$\underbrace{UcorSDtoChol\left\{\begin{bmatrix}\text{diff\_eta1} & 0\\ \text{diff\_eta2\_eta1} & \text{diff\_eta2}\end{bmatrix}\right\}}_{\underset{\text{DIFFUSION}}{\mathbf{G}}}\underbrace{\mathrm{d}\begin{bmatrix}W_1\\ W_2\end{bmatrix}(t)}_{\mathrm{d}\mathbf{W}(t)}$$

Observations:

$$\underbrace{\begin{bmatrix}\text{Y1}\\ \text{Y2}\end{bmatrix}(t)}_{\mathbf{Y}(t)} = \underbrace{\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}}_{\underset{\text{LAMBDA}}{\boldsymbol{\Lambda}}}\underbrace{\begin{bmatrix}\text{eta1}\\ \text{eta2}\end{bmatrix}(t)}_{\boldsymbol{\eta}(t)} + \underbrace{\begin{bmatrix}0\\ 0\end{bmatrix}}_{\underset{\text{MANIFESTMEANS}}{\boldsymbol{\tau}}} \quad +$$

Observation noise:

$$\underbrace{\begin{bmatrix}\text{mvarY1} & 0\\ 0 & \text{mvarY2}\end{bmatrix}}_{\underset{\text{MANIFESTVAR}}{\boldsymbol{\Theta}}}\underbrace{\begin{bmatrix}\epsilon_1\\ \epsilon_2\end{bmatrix}(t)}_{\boldsymbol{\epsilon}(t)}$$

System noise distribution per time step: $\Delta[W_{j\in[1,2]}](t-u) \sim \text{N}(0, t-u)$

Observation noise distribution: $\left[\epsilon_{j\in[1,2]}\right](t) \sim \text{N}(0,1)$

Note: *UcorSDtoChol* converts lower tri matrix of standard deviations and unconstrained correlations to Cholesky factor, *UcorSDtoCov* = transposed cross product of UcorSDtoChol, to give covariance, See Driver & Voelkle (2018) p11. Individual specific notation (subscript i) only shown for subject parameter distribution – pop. means shown elsewhere.

```
head(model$pars,4)
```

```
   matrix row col    param value  transform indvarying sdscale TI1_effect
1 TOMEANS   1   1 T0m_eta1    NA 10 * param       TRUE       1      FALSE
2 TOMEANS   2   1 T0m_eta2    NA 10 * param       TRUE       1      FALSE
3  LAMBDA   1   1     <NA>     1       <NA>      FALSE      NA      FALSE
4  LAMBDA   1   2     <NA>     0       <NA>      FALSE      NA      FALSE
```

By default, intercept related model parameters (T0MEANS, MANIFESTMEANS, CINT) are set to individually varying using a correlated random effects formulation, while other parameters are fixed across subjects. In contrast to this, when time independent predictors are included (time invariant covariates), the default (adjustable using the `tipredDefault` argument to ctModel) that they affect all free subject level parameters. One may modify the output model to further restrict, or allow, between subject differences (set some parameters to not vary over individuals), alter the transformation used to set the prior / bounds, or restrict which effects of time independent predictors to estimate – see the section on adjusting model specification for details on this.

Priors are not included by default but for those who wish to use Bayesian estimation approaches, they can be switched on during model fitting using the `priors=TRUE` argument. For the default of maximum likelihood the information in this text regarding priors is obviously less relevant, however some awareness may be useful as a) priors are always used during an initial rough estimation, and b) although the prior probability is not relevant, some parameters (such as standard deviations) rely on transformations to ensure they stay within a particular range.

1.7 Model fitting

Once model specification is complete, the model is fit to the data using the `ctStanFit` function as shown in the following example. Various options are available for fitting, from a maximum likelihood approach using optimization, to a fully Bayesian approach via Stan's Hamiltonian Monte Carlo sampler. For these examples we will use optimization, though for cases with many random effects on non-intercept parameters, or missing time independent predictors, optimization may not always provide the most appropriate results and should be interpreted with care.

For a classic maximum likelihood approach, optimization may be used with the argument (set by default) `priors=FALSE` to disable the priors. By default, when optimization concludes, samples are drawn from the parameter covariance matrix implied by the Hessian to allow for inference and uncertainty plotting on all aspects of the model (e.g., parameter interactions such as the implied regression effect for a given time interval). Improvements on this maximum a posteriori, Hessian based sampling approach can be made by requesting importance sampling (see the `optimcontrol` argument of the `?ctStanFit` help, which is somewhat slower but generally still faster than full sampling using Stan's Hamiltonian Monte Carlo (HMC) – the caveats mentioned above still apply however, and importance sampling may not work well with many parameters. When importance sampling is ineffective, this will generally be indicated by errors during the sampling process and possibly an inability to converge. Stan's HMC is in general the most robust solution, and can be obtained by setting `optimize=FALSE` and `priors=TRUE`. Note that if random effects are specified when optimization is used, between subject differences are integrated over. Estimates of the subject specific parameters can be obtained via the `ctStanKalman` function.

When `optimize=FALSE`, Stan's Hamiltonian Monte Carlo sampler is used. Depending on the data, model, and number of iterations requested, this can take anywhere from a few minutes to days. Im-

portant arguments when using this approach are `chains` and `iterations` which allow specification of the number of sampling chains, and iterations per chain. Three chains is a reasonable minimum to support convergence checking, and current experience suggests 300 iterations is often enough to get an idea of what is going on – more is very likely necessary for robust inference, but how much more is highly dependent on the data and model. To examine progress while sampling, the `plot=TRUE` argument may be used. In some cases, extended warmup / adaptation times can be avoided (at cost of reduced computational efficiency during the actual sampling) by reducing the maximum treedepth using the `control` argument.

```
fit<-ctStanFit(datalong = ctstantestdat, ctstanmodel = model, priors=TRUE)
```

1.8 Summary

After fitting, the summary function may be used on the fit object, which returns details regarding the population mean parameters, population standard deviation parameters, population correlations, and the effect parameters of time independent predictors. Additionally, the summary function outputs a range of matrices regarding correlations between subject level parameters. `rawpopcorr_means` reports the posterior mean of the correlation between raw (not yet transformed from the standard normal scale) parameters. `rawpopcorr_sd` reports the standard deviation of these parameters.

```
summary(fit, timeinterval = 1)
```

In the summary output, the free population mean parameters under `$popmeans` are likely one of the main points of interest. They are returned in the same form that they are input to `ctModel` - that is, covariance matrix related parameters are in the form of either standard deviations or a transformed correlation parameter. Because the latter is difficult to interpret, various parameter matrices are also returned in the `$parmatrices` section of the summary. The discrete time matrices reported here (prefixed by `dt`, unless a discrete time model was estimated in the first place) are by default from a time interval of 1, but this can be changed. Asymptotic matrices – those for a time interval of infinity – are also output in some cases, and prefixed by `asym`. Covariance related matrices are reported in covariance form, except where the suffix `cor` is added to indicate correlations.

It is possible to represent the results for the within subject model using the Latex representation, running `ctModelLatex(fit)` will show this.

The function `ctStanContinuousPars` can be used to return the population level parameter matrices in matrix form. Here we return the 97.5 quantile for each of the elements in the matrices:

```
ctStanContinuousPars(fit,calcfunc = quantile, calcfuncargs = list(probs=.975))
```

1.9 Plotting

The plot function outputs a sequence of plots, all generated by specific functions. The name of the specific function appears in a message in the R console, checking the help for each specific function and running them separately will allow more customization of plots. Depending on model specifications and fitting, not all plots are available. Some of the plots, such as the trace, density,

and interval, are generated by the relevant rstan function and are hopefully self explanatory. The plots specific to the hierarchical continuous time dynamic model are as follows:

```
try({
ctStanDiscretePars(fit, plot=TRUE, indices = 'CR')
})
```

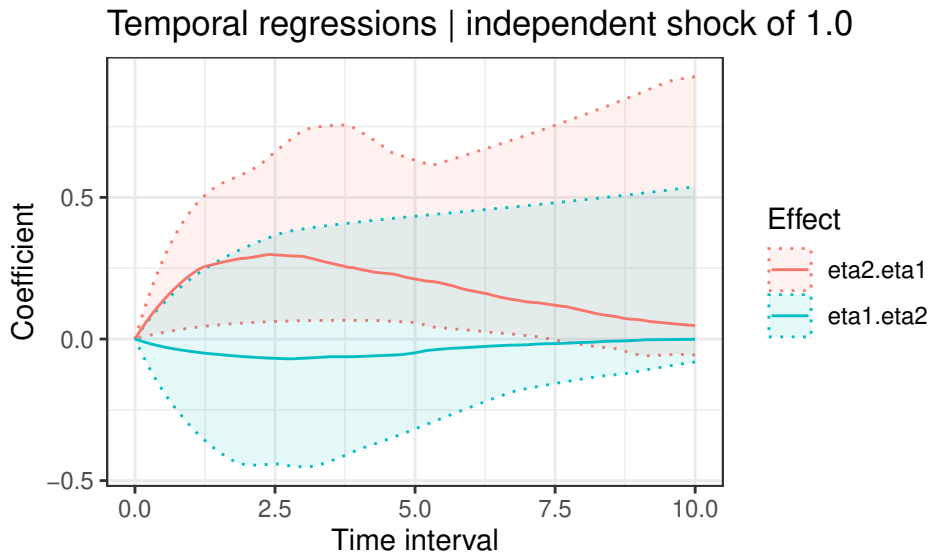

Figure 2: Discrete-time cross-effect dynamics of the estimated system for a range of time intervals, with 95% credible intervals.

Figure 2 shows the dynamic regression coefficients (between latent states at different time points) that are implied by the model for particular time intervals, as well as the uncertainty (default is 95% credible interval) of these coefficients. In this case the estimates are of the cross regression effects, obtained by sampling from all subjects data, but specific subjects, as well as specific indices of the effects (e.g., indices = 'AR' or indices = rbind(c(2,1))' can be specified. The mapping of subject id to the internal sequential integer representation can be found via `fit$setup$idmap` .

The relation between posteriors and priors for variables of interest can also be plotted as follows – note that the `rows` argument of the shown code is not necessary, but if only particular parameter plots are desired the rows corresponding to specific parameters of `fit$setup$popsetup` can be specified.

```
ctStanPlotPost(obj = fit, rows=3)
```

Shown in Figure 3 are approximate density plots based on the post-warmup samples drawn. For each parameter that has individual variation specified, three plots are shown. These are: 1) the population mean posterior compared to the prior; 2) the posterior (available only with sampling) versus prior distribution of subject level parameters along with the population mean prior; 3) and then the population standard deviation posterior compared to the prior.

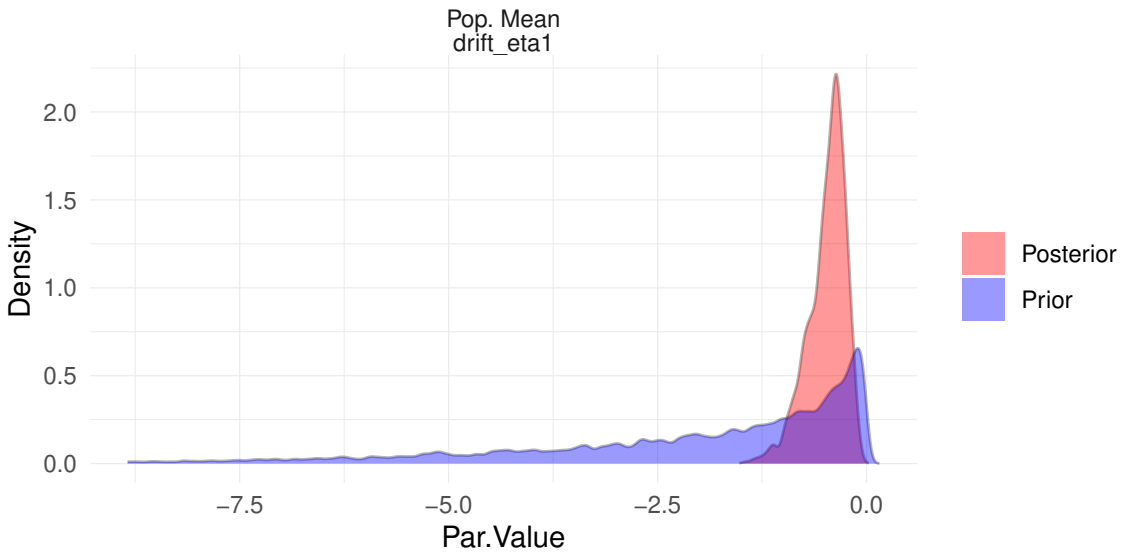

Figure 3: Prior and posterior densities relevant to the first variables manifest intercept.

1.10 Model prediction plots

One means of assessing model performance is to view plots of the observed time series alongside the model predicted time series. ctsem includes functionality to output prior (based on all prior observations) and updated (based on all prior and current observations) expectations and covariances from the Kalman filter, based on specific subjects models. Examples of such are depicted in Figure 4, where we see observed, predicted, and smoothed scores for a selected subject from our sample. If we wanted to predict unobserved states in the future, we would need only to specify the appropriate timerange (prediction into earlier times is possible but makes little sense unless the model is restricted to stationarity). For help with these plots, see `?ctKalman` and `?ctKalmanPlot` (arguments for the latter are passed via `ctKalman`, as below).

```
ctKalman(fit, subjects=c(2,4,5),  kalmanvec=c('y', 'yprior'),
  plot=TRUE, timestep=.01)
```

1.11 Time independent predictor effect plots

Because time independent predictors give a linear effect *prior* to any necessary transformations, the effects necessarily become non-linear when applied to bounded parameters, which can make them difficult to conceptualise. To aid with this, a visual summary of the full range of effects can be seen using the `ctStanTIpredeffects` function, as follows:

```
ctStanTIpredeffects(fit, plot = TRUE, whichpars=c('dtDRIFT[2,1]','CINT'),
 timeinterval = .5, whichTIpreds = 1, includeMeanUncertainty = TRUE)
```

Figure 5 shows how the expectation for an individuals parameter value is likely to change depending on the value they have for the time independent predictor specified. In this example, the discrete time drift effects for a time interval of 0.5, as well as the continuous intercepts parameters, are

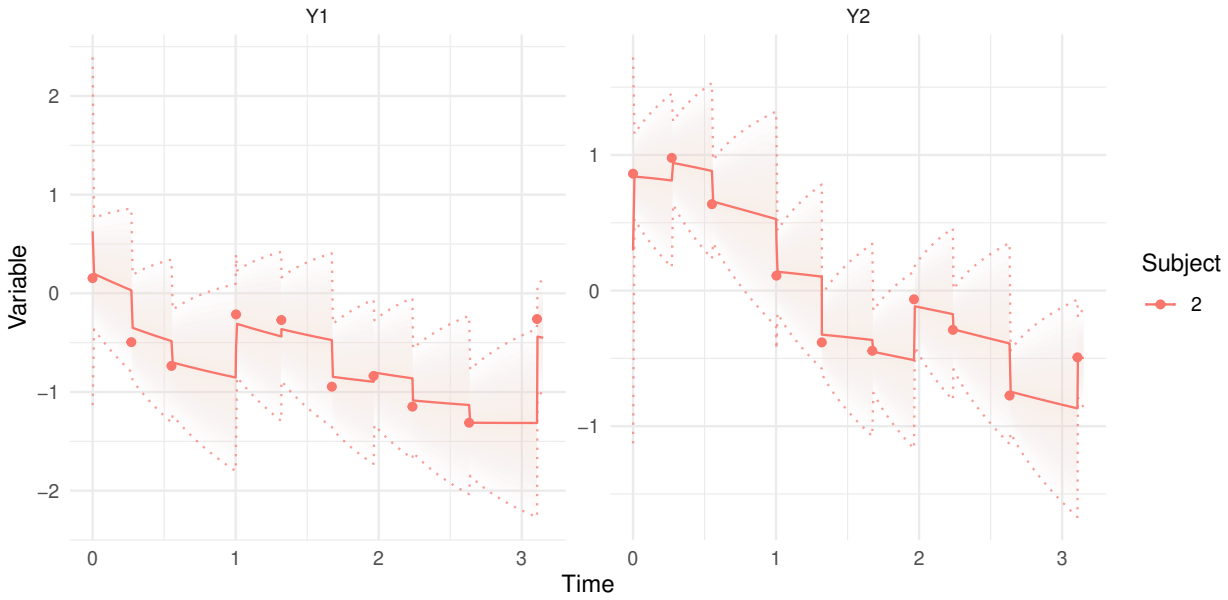

Figure 4: Predictions for three subjects over two processes. Uncertainty shown is a 95% credible interval comprising both process and measurement error.

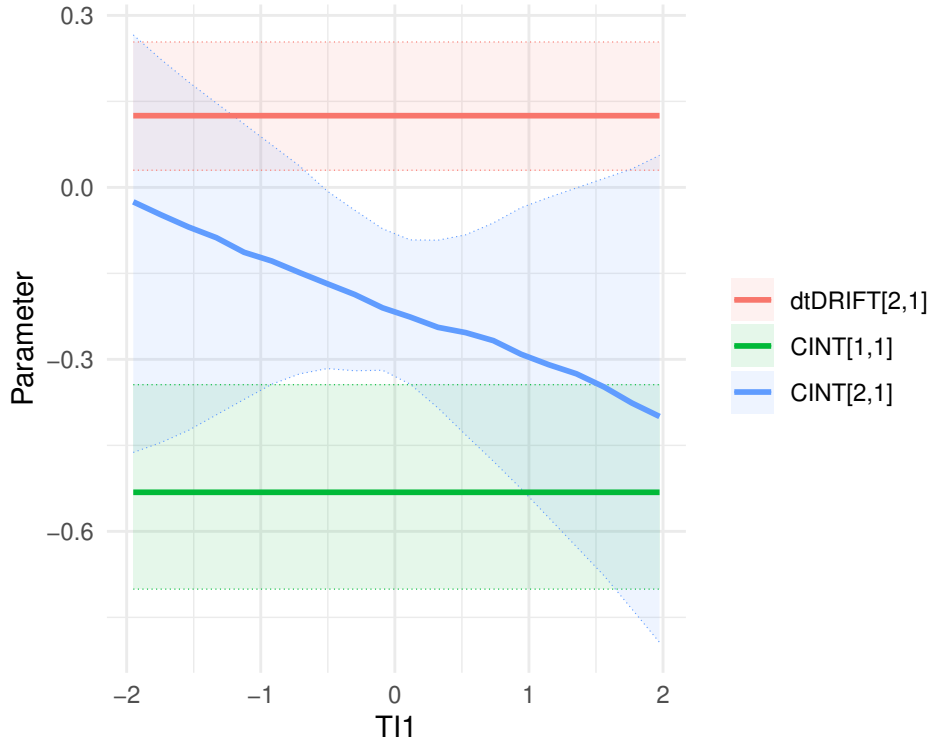

Figure 5: Expectations for individuals parameter values change depending on their score on time independent predictors.

shown. Only one of these has a non-zero effect, because the others were not included in our model. Multiple predictors can be specified, and the combined effect based on observed predictor combinations will be shown, but this will likely make no sense unless there is a deterministic relation between the two – this can be useful for including linear and quadratic effects, for instance, wherein the first predictor is linear, and the second quadratic. The nsamples and nsubjects parameters specify how many different parameter samples, and predictor values, are used – higher values take longer to compute, but give smoother / more accurate plots.

2 Additional details

2.1 Detailed model specification

While the defaults are hopefully useful in many cases, they should not be blindly relied upon. Plotting the original prior implied by the model, making a change to the transform, and plotting the resulting prior, are shown here – in this case we will adjust the prior for the auto effect of our first latent process, captured by row 1 and column 1 of the DRIFT matrix, to also allow positive values, implying an explosive process wherein a change in one direction promotes further change in that direction. To achieve this, we change from the default `-(2 * log1p(exp(2 * param)))` (a scaled 'softplus' function that works well for parameters that need either an upper or lower boundary), to a simpler transform wherein we just shift and scale the raw, standard normal, parameter. In this case we also set a negative offset because we still believe negative values are more likely for the drift auto-effect, and we also start by allowing for unobserved individual differences by adjusting the `$indvarying` column of the parameter list (in order to visualise differences in the resulting distributions). One consideration when altering priors in this manner is that the starting values for sampling are taken from around the median of the distribution. The following code shows two approaches to changing a parameter specification. The first is done after initial model specification by directly editing the model object, the second is done during model specification by specifying one or more of parameter name, transformation, TRUE or FALSE for random effects, a positive numeric for the prior standard deviation of the random effects relative to the prior for the mean, then TRUE or FALSE for any time independent predictors. These elements are input in an ordered sequence separated by the | character, such that in order to specify individual variation (for example), the parameter name and transform must also be specified, but any subsequent elements will assume default values. The example only sets the first two elements, with the rest assuming defaults.

```
#Manual change after model specification
model$pars$indvarying[7] <- TRUE
p <- plot(model, rows=7,rawpopsd=1, plot=FALSE)
print(p[[1]] + ggplot2::theme(legend.position = "none"))
model$pars$transform[7]<- '2 * param -1'

#Change during model specification
model<-ctModel(type='stanct',
        DRIFT = matrix(c('drift_eta1_eta1 | 2 * param -1 | TRUE',
          'dr21','dr12','dr22'),2,2),
        latentNames=c('eta1','eta2'),
        manifestNames=c('Y1','Y2'),
        TDpredNames='TD1',
```

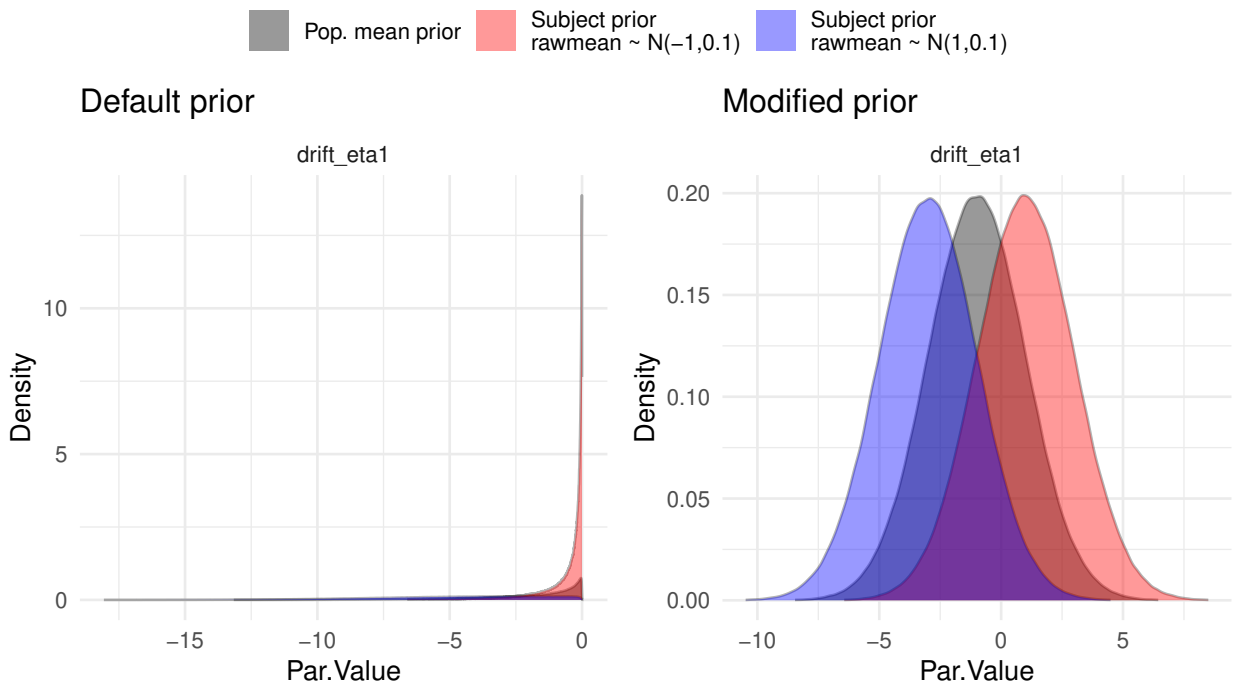

Figure 6: Prior distribution density plots.

```
        TIpredNames=c('TI1'),
        LAMBDA=diag(2))

plot(model, rows=7,rawpopsd=1)
```

Figure 6 shows the prior distribution for the population mean of DRIFT[1,1] in black, as well as two possible priors for the subject level parameters, conditional on our specified raw population standard deviation of 1. The blue prior results from assuming the population mean is one standard deviation lower than the mean of its prior, and the red one standard deviation higher.

In addition to adjusting the prior for the population mean, the prior for the extent of individual variation around that mean can also be adjusted. Amongst other circumstances, this prior may need to be reduced when limited time points are available, to ensure adequate regularisation. Here we change the scaling factor of the individual variation for all parameters, from 1.0 to 3.0, and demonstrate the effect of this using the previously adjusted auto effect. In this case, we do not fix the `popsd` when plotting, which now gives the distribution of individual variation over all possible values for the population (pop) sd parameter – the marginal distribution of Figure 7 has a very different shape to the conditional distribution of Figure 6, but the effect of the change in `sdscale` parameter could be seen in both cases.

```
plot(model, rows=7)
model$pars$sdscale<- .1
plot(model, rows=7)
```

It can be helpful to completely eliminate individual variation in some parameters, particularly since unnecessary between subject effects will slow sampling and hinder appropriate regularization, but

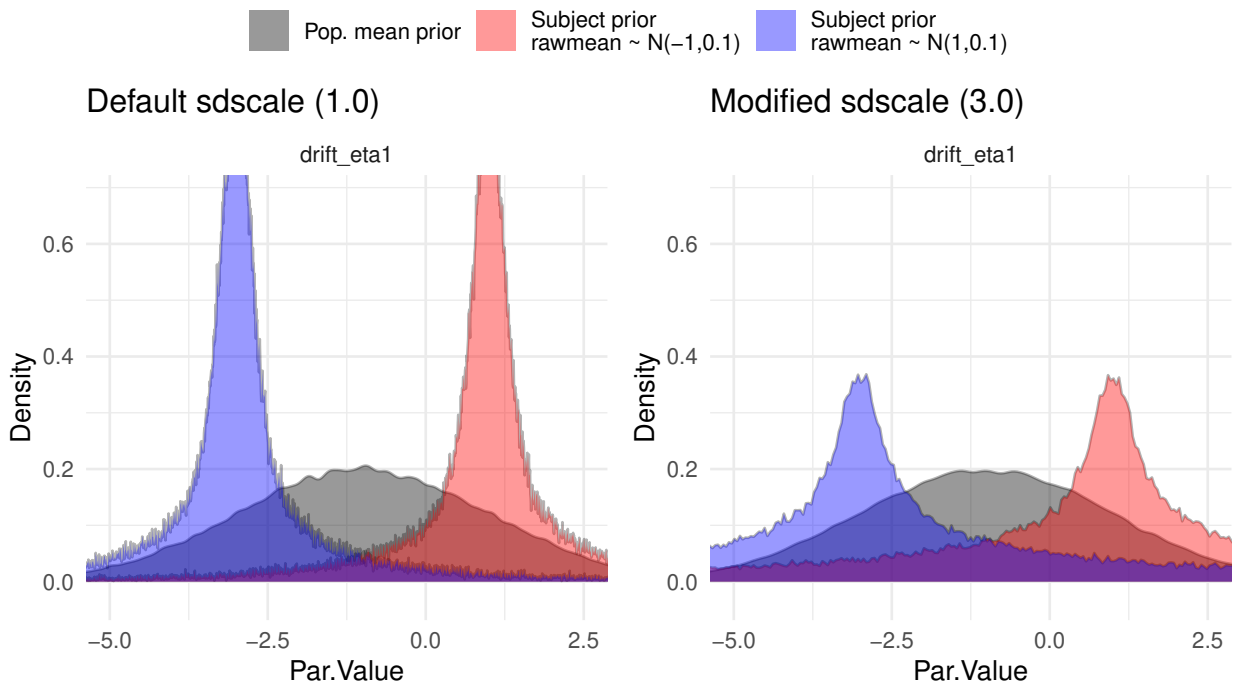

Figure 7: Prior distribution density plots of auto-effects, with default (left) and adjusted (right) scale parameter for population standard deviation.

be aware of the many parameter dependencies in these models – restricting one parameter may lead to genuine variation in the restricted parameter expressing itself elsewhere. Here we only allow for individual variation in the DRIFT and MANIFESTMEANS parameters.

```
model$pars$indvarying[ !(model$pars$matrix %in% c('DRIFT','MANIFESTMEANS'))] <- FALSE
model$pars$indvarying[ (model$pars$matrix %in% c('DRIFT','MANIFESTMEANS'))] <- TRUE
```

Also similarly restrict which parameters to include time independent predictor effects for. In general, when random effects are allowed, population mean estimates should not depend greatly on whether covariate effects are included or not. If random effects are restricted (which they are by default) for computational or theoretical reasons, population estimates will often be more dependent on the covariates included and the parameters that are affected. Here, we first restrict the tipredeffects on all parameters, and free them only for the drift parameters.

```
model$pars[,c('TI1_effect','TI2_effect','TI3_effect')] <- FALSE
model$pars[model$pars$matrix == 'DRIFT',
  c('TI1_effect','TI2_effect','TI3_effect')] <- TRUE
```

An alternative approach to specifying the tipredeffects is in the initial model specification, using the '|' separator notation. The following shows an example where the DRIFT[1,1] parameter is influenced by covariates TI1 and TI3, DRIFT[1,2] is influenced by TI2, and there are no other time independent predictor effects (because `tipredDefault=FALSE`)

```
model<-ctModel(type='stanct', tipredDefault = FALSE,
        DRIFT = matrix(c(
          'drift_eta1_eta1||||TI1,TI3','dr21||||TI2',
          'dr12','dr22'),2,2),
        latentNames=c('eta1','eta2'),
        manifestNames=c('Y1','Y2'),
        TDpredNames='TD1',
        TIpredNames=c('TI1'),
        LAMBDA=diag(2))
```

2.2 Accessing Stan model code

For diagnosing problems or modifying the model in ways not achievable via the ctsem model specification, one can use ctsem to generate the Stan code and then work directly with that, simply by specifying the argument `fit=FALSE` to the `ctStanFit` function, and accessing the `$stanmodeltext` subobject. Any altered code can be passed back into `ctStanFit` by using the `stanmodeltext` argument, which can be convenient for setting up the data in particular.

2.3 Using Rstan functions

The standard rstan output functions such as summary and extract are also available whenever Stan's sampler is used, and the shinystan package provides an excellent browser based interface. The stan fit object is stored under the `$stanfit` subobject of the `ctStanFit` output. The parameters which are likely to be of most interest in the output are prefixed by `pop_` for pop (population) mean, and `popsd` for pop standard deviation. Any `pop` parameters are returned in the form of the continuous time matrix equations (or discrete time when estimated as such). Subject specific parameters are denoted by the matrix they are from, then the first index represents the subject id, followed by standard matrix notation. For example, the 2nd row and 1st column of the DRIFT matrix for subject 8 is `DRIFT[8,2,1]`. Parameters in such matrices are returned in the form used for internal calculations – that is, variance covariance matrices are returned as such, rather than the lower-triangular standard deviation and correlation matrices required for input.

2.4 Oscillating, single subject example - sunspots data

To demonstrate fitting more complicated higher order processes and oscillations, we use the sunspots data available within R, which has previously been fit by various authors including Tómasson (2013). We have used the same CARMA(2,1) model and obtained similar estimates. For more on CARMA modelling and more detail on this sunspots example, see Oud et al. (2018).

```
#get data
sunspots<-sunspot.year
sunspots<-sunspots[50: (length(sunspots) - (1988-1924))]
id <- 1
time <- 1749:1924
datalong <- cbind(id, time, sunspots)

#setup model
```

```
ssmodel <- ctModel(type='stanct',
  manifestNames='sunspots',
  latentNames=c('ss_level', 'ss_velocity'),
  LAMBDA=c( 1, 'ma1| log(1+(exp(param)))'),
  DRIFT=c(0, 1,
    'a21 | -log(1+exp(param))','a22'),
  MANIFESTMEANS=c('m1|param * 10 + 44'),
  MANIFESTVAR=diag(0,1), #As per original spec
  CINT=0,
  DIFFUSION=c(0, 0,
    0, "diffusion"))
```

```
     [,1] [,2]
[1,]  "1" "ma1| log(1+(exp(param)))"
     [,1]                        [,2]
[1,]                          "0"   "1"
[2,] "a21 | -log(1+exp(param))" "a22"
     [,1] [,2]
[1,]  "0"         "0"
[2,]  "0" "diffusion"
     [,1]
[1,] "m1|param * 10 + 44"
     [,1]
[1,]  "0"
[2,]  "0"
```

```
#fit
ssfit <- ctStanFit(datalong, ssmodel,
  iter=300, chains=2, optimize=FALSE, priors=TRUE)

#output
summary(ssfit)$popmeans
```

```
                  mean     sd   2.5%    50%  97.5% n_eff  Rhat
T0m_ss_level    22.584 6.2262 10.770 22.994 33.942 133.4 1.000
T0m_ss_velocity 15.489 7.1025  2.658 15.377 29.401 192.8 1.004
ma1              0.723 0.2143  0.346  0.711  1.193 107.7 0.995
a21             -0.352 0.0461 -0.451 -0.348 -0.270 137.6 0.998
a22             -0.322 0.1032 -0.571 -0.309 -0.168 112.6 0.995
diffusion       15.305 3.2402 10.109 14.852 23.188  82.3 0.998
m1              46.604 3.0051 40.773 46.310 53.791 181.1 1.007
```

2.5 Non-linearities

The dynamic model that arises normally out of a typical ctsem model specification is linear, as the dynamic system and measurement matrices (DRIFT, MANIFESTVAR, etc.) do not vary over time. This can be changed by making certain elements of the matrices depend on the current state of the system. This is done by setting the element to a character string containing the reference to 'state', and the specific state. In a simple univariate, first order linear system, the DRIFT matrix would be 1×1 and contain a character string such as 'drift11', indicating we want to estimate the

parameter called 'drift11' for the DRIFT matrix. This parameter could however be made state dependent by specifying the DRIFT matrix element as a function of one or more parameters and latent states. Careful thought regarding the required model structure and restrictions is necessary. Here, we modify the oscillatorysunspots example from earlier, such that the oscillation damping factor is no longer constant but depends on the current level of sunspot activity. It is possible to reference the current states of the process by using the appropriate latent variable name, and additional parameters may be included so long as they are also specified in the PARS model element. Whenever a latent variable or additional parameter is referenced, additional formulas that are valid in the Stan language are also possible to use – `"exp(ss_level)"` takes the exponent of the current state of the sunspot level latent variable, and would ensure positivity of the result, for instance. In the following example `-log1p_exp` is used to ensure negativity using a more linear transform than the exponential.

```
sunspots<-sunspot.year
sunspots<-sunspots[50: (length(sunspots) - (1988-1924))]
id <- 1
time <- 1749:1924
datalong <- data.frame(id, time, sunspots)

m <- ctModel(type='stanct',
  manifestNames='sunspots',
  latentNames=c('ss_level', 'ss_velocity'),
  LAMBDA=c( 1, 'ma1|log(1+exp(param))'),
  DRIFT=c(0, 1,
    '-log1p_exp(freqintercept + freqbylevel *  ss_level)','a22'),
  MANIFESTMEANS=c('m1|param * 10 + 44'),
  MANIFESTVAR=diag(0,1), #As per original spec
  CINT=0,
  DIFFUSION=c(0, 0,
    0, "diffusion"),
  PARS=c('freqintercept', 'freqbylevel'))

ssfitnl <- ctStanFit(datalong, m)
```

2.6 Population standard deviations - understanding the transforms

This section is intended as a helper to those trying to work through the various transformations found in the model. Internally, we sample parameters that we refer to as the 'raw' parameters – these parameters have no bounds and are typically drawn from normal distributions. Both raw population mean and subject specific deviation parameters are drawn from normal(0, 1) distributions. Depending on the specific parameter, various transformations may be applied to set appropriate bounds and priors. The raw population standard deviation for these raw parameters is sampled (by default) from a normal(0, 1) distribution called `rawpopsdbase`, which is by default transformed via an exponential function – this ensures the parameters are positive and the prior for the standard deviation is a lognormal distribution. This distribution can be altered via the model subobjects `rawpopsdbase`, `rawpopsdbaselowerbound`, and `rawpopsdtransform`. This distribution can also be scaled on a per parameter basis by the `sdscale` multiplier in the model specification, which defaults to 1. The following script shows a didactic sequence of sampling and transformation for a model with a single parameter, the auto effect of the drift matrix, and 3 subjects. Although we sample the priors themselves here, this is merely to reflect the prior and enable understanding and plotting. Note also that because we are only displaying the procedure for a single parameter here, we simplify

things somewhat by avoiding calculations to determine the square root of the population covariance matrix – with only one individually varying parameter, it is simply the standard deviation.

```
#set plotting parameters
par(mfrow=c(2,2), lwd=3, yaxs='i', mgp=c(1.8,.5,0),
  mar=c(3,3,3,1)+.1)
bw=.03

n <- 999999 #number of samples to draw to from prior for plotting purposes
nsubjects <- 4 #number of subjects

#parameter specific transform
tform <- function(x) -log(exp(-1.5 * x) + 1) #default drift auto effect transform

#raw pop sd transform
sdscale <- 1 #default
rawsdtform <- function(x) exp(x * 2 -2) * sdscale #default

#sd approximation function
sdapprox <- function(means,sds,tform) {
  for(i in 1:length(means)){
    sds[i] <- ((tform(means[i]+sds[i]*3) - tform(means[i]-sds[i]*3))/6 +
      (tform(means[i]+sds[i]) - tform(means[i]-sds[i]))/2) /2
  }
  return(sds)
}

#raw population mean parameters
rawpopmeans_prior <- rnorm(n, 0, 1) #prior distribution for rawpopmeans
rawpopmeans_sample <- -.3 #hypothetical sample
sdscale <- 1 #default

#population mean parameters after parameter specific transform
popmeans_prior <- tform(rawpopmeans_prior)
popmeans_sample <- tform(rawpopmeans_sample)

#plot pop means
plot(density(rawpopmeans_prior), ylim=c(0,1), xlim=c(-5,2),
  xlab='Parameter value', main='Population means')
points(density(popmeans_prior, bw=bw),col=2,type='l')
segments(y0=0,y1=.5,x0=c(rawpopmeans_sample,popmeans_sample),lty=3,col=1:2)
legend('topleft',c('Raw pop. mean prior', 'Pop. mean prior',
  'Raw pop. mean sample', 'Pop. mean sample'),lty=c(1,1,3,3), col=1:2, bty='n')

#population standard deviation parameters
rawpopsd_prior <- rawsdtform(rnorm(n, 0, 1)) #raw population sd prior

popsd_prior <- sdapprox(rawpopmeans_prior,rawpopsd_prior,tform)

#sample population standard deviation posterior
rawpopsd_sample <- rawsdtform(.9) #hypothetical sample
popsd_sample <- sdapprox(means=rawpopmeans_sample,  #transform sample to actual pop sd
  sds=rawpopsd_sample,tform=tform)
```

```
#plot pop sd
plot(density(rawpopsd_prior,from=-.2,to=10,na.rm=TRUE, bw=bw), xlab='Parameter value',
  xlim=c(-.1,3), ylim=c(0,2), main='Population sd')
points(density(popsd_prior,from=-.2,to=10,na.rm=TRUE, bw=bw),type='l', col=2)
segments(y0=0,y1=1,x0=c(rawpopsd_sample, popsd_sample), col=1:2,lty=3)
legend('topright',c('Raw pop. sd prior','Pop. sd prior',
  'Raw pop. sd sample','Pop. sd sample'), col=1:2, lty=c(1,1,3,3),bty='n')

#individual level parameters

#marginal individual level parameters (given all possible values for mean and sd)
rawindparams_margprior <- rawpopmeans_prior + rawpopsd_prior * rnorm(n, 0, 1)
indparams_margprior <- tform(rawindparams_margprior)

plot(density(rawindparams_margprior,from=-10,to=10,bw=bw), xlab='Parameter value',
  xlim=c(-5,2), ylim=c(0,1), main='Marginal dist. individual parameters')
points(density(indparams_margprior,from=-10,to=.2,bw=bw),type='l',col=2)
legend('topleft',c('Raw individual parameters prior','Individual parameters prior'),
  col=1:2,lty=1,bty='n')


#conditional individual level parameters (given sampled values for mean and sd)
rawindparams_condprior<- rawpopmeans_sample + rawpopsd_sample * rnorm(n,0,1)
rawindparams_condsample<- rawpopmeans_sample + rawpopsd_sample * rnorm(nsubjects,0,1)
indparams_condprior<- tform(rawindparams_condprior)
indparams_condsample<- tform(rawindparams_condsample)


plot(density(rawindparams_condprior), xlab='Parameter value', xlim=c(-5,2),
  ylim=c(0,1), main='Conditional dist. individual parameters')
points(density(indparams_condprior),type='l',col=2)
segments(y0=0,y1=.5,x0=c(rawindparams_condsample, indparams_condsample),
  col=rep(1:2,each=nsubjects),lty=3, lwd=2)
legend('topleft',c('Raw ind. pars. prior','Ind. pars. prior',
  'Raw ind. pars. samples','Ind. pars. samples'), col=1:2, lty=c(1,1,3,3),bty='n')
```

In the top left of Figure 8, we can see the prior distribution of population means for, in this case, a diagonal (auto effect) of the drift matrix. The prior for the raw population distribution is a standard normal, while for the actual population distribution it is definitely not normal. We draw a hypothetical sample from the raw distribution, and show the resulting transformed value. To the right, the prior distribution of the raw population standard deviation is shown. This raw distribution is the same for all parameter types, but the resulting prior distribution of population standand deviations is also dependent on the parameter specific transform (although in this particular case the raw and actual population sd priors are almost the same). This dependency is most easily understood if one considers the case where the parameter specific transformation simply multiplied the raw parameter by 2 – if we sampled a raw population sd of 1.5, the actual population sd sample would be 3.0. With nonlinear transformations, the dependency is not so easily calculated, and we use a sigma point approximation (Julier & Uhlmann, 1997), as shown in the code, when it is necessary to plot or summarise the population sd. The lower left plot shows the prior distribution for individual level parameters, marginalising over the priors for population means and standard deviations. This plot is very similar to the means plot directly above it, just somewhat more spread

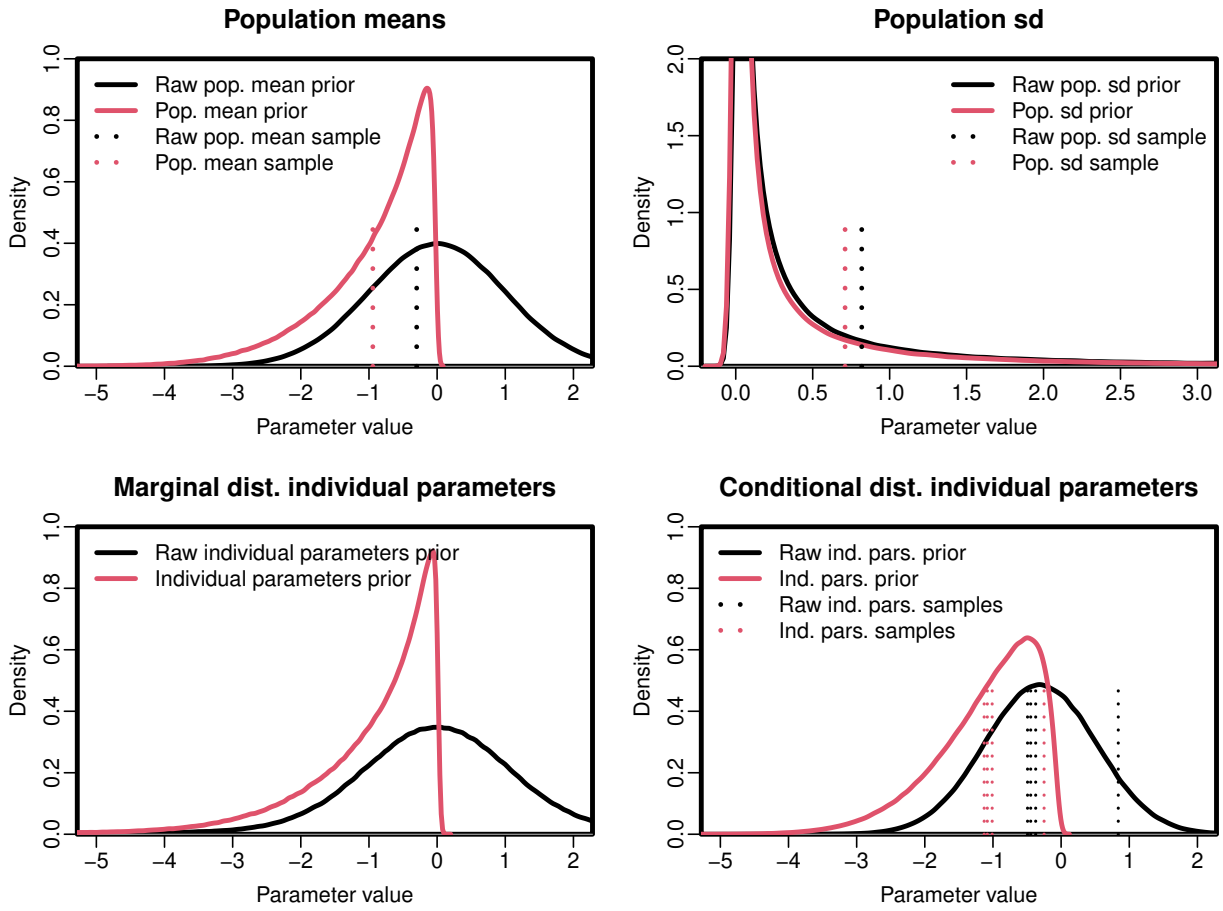

Figure 8: Depiction of the prior distributions and sampling process through which individual specific parameters are determined. Note that the population sd is strictly positive, however the density plots involve some smoothing."

out due to the additional variation included. Things get more interesting when we look at the lower right plot – here, we see the prior distributions for individual level parameters, conditional on the values sampled in the top row of plots. Along with the prior distribution, in this lower right plot we also draw samples for 4 subjects, showing both the raw individual parameter, and the individual parameter after the necessary transforms.

3 Conclusion

With this work, we have described the basics of the hierarchical continuous time dynamic model, and provided detailed discussion on the usage of the R package ctsem (Driver et al., 2017) for fitting such models to data. While the approach is necessarily somewhat complex, we believe it offers many interesting possibilities for understanding within-subject dynamics and their relation to between-subject differences, and hope that the overview of the software provided here encourages new and interesting applications of the model..

References

Carpenter, B., Gelman, A., Hoffman, M. D., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M., Guo, J., Li, P., & Riddell, A. (2017). Stan: A probabilistic programming language. *Journal of Statistical Software*, *76*(1). https://doi.org/10.18637/jss.v076.i01

Driver, C. C., Oud, J. H. L., & Voelkle, M. C. (2017). Continuous time structural equation modeling with r package ctsem. *Journal of Statistical Software*, *77*(5). https://doi.org/10.18637/jss.v077.i05

Driver, C. C., & Voelkle, M. C. (2018a). Hierarchical Bayesian continuous time dynamic modeling. *Psychological methods*, *23*(4), 774.

Driver, C. C., & Voelkle, M. C. (2018b). Understanding the time course of interventions with continuous time dynamic models. In K. van Montfort, J. H. L. Oud & M. C. Voelkle (Eds.), *Continuous time modeling in the behavioral and related sciences*. Springer International Publishing. //www.springer.com/de/book/9783319772189

Gelman, A., Carlin, J. B., Stern, H. S., & Rubin, D. B. (2014). *Bayesian data analysis* (Vol. 2). Chapman & Hall/CRC Boca Raton, FL, USA. http://amstat.tandfonline.com/doi/full/10.1080/01621459.2014.963405

Julier, S. J., & Uhlmann, J. K. (1997). A New Extension of the Kalman Filter to Nonlinear Systems, 182–193.

Neale, M. C., Hunter, M. D., Pritikin, J. N., Zahery, M., Brick, T. R., Kirkpatrick, R. M., Estabrook, R., Bates, T. C., Maes, H. H., & Boker, S. M. (2016). OpenMx 2.0: Extended structural equation and statistical modeling. *Psychometrika*, *81*(2), 535–549. https://doi.org/10.1007/s11336-014-9435-8

Oud, J. H. L., & Jansen, R. A. R. G. (2000). Continuous time state space modeling of panel data by means of SEM. *Psychometrika*, *65*(2), 199–215. https://doi.org/10.1007/BF02294374

Oud, J. H. L., Voelkle, M. C., & Driver, C. C. (2018). SEM based CARMA time series models for arbitrary N. *Multivariate Behavioral Research*, *53*, 36–56. https://doi.org/10.1080/00273171.2017.1383224

R Core Team. (2014). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing. http://www.R-project.org/

Singer, H. (1993). Continuous-time dynamical systems with sampled data, errors of measurement and unobserved components [00046]. *Journal of Time Series Analysis*, *14*(5), 527–545. https://doi.org/10.1111/j.1467-9892.1993.tb00162.x

Stan Development Team. (n.d.). *RStan: The R interface to Stan* (Version 2.18.1). http://mc-stan.org

Tómasson, H. (2013). Some computational aspects of Gaussian CARMA modelling. *Statistics and Computing*, *25*(2), 375–387. https://doi.org/10.1007/s11222-013-9438-9

Voelkle, M. C., & Oud, J. H. L. (2013). Continuous time modelling with individually varying time intervals for oscillating and non-oscillating processes. *British Journal of Mathematical and Statistical Psychology*, *66*(1), 103–126. https://doi.org/10.1111/j.2044-8317.2012.02043.x