Introduction to Hierarchical Continuous Time Dynamic Modelling With ctsem

Charles C. Driver
Max Planck Institute for Human Development

Manuel C. Voelkle
Humboldt University Berlin
Max Planck Institute for Human Development

Abstract

ctsem allows for specification and fitting of a range of continuous and discrete time dynamic models with stochastic system noise. The models may include multiple indicators (dynamic factor analysis), multiple, interrelated, potentially higher order processes, and time dependent (varying within subject) and time independent (not varying within subject) covariates. Classic longitudinal models like latent growth curves and latent change score models are also possible. Version 1 of ctsem provided structural equation model based functionality by linking to the OpenMx software, allowing mixed effects models (random means but fixed regression and variance parameters) for multiple subjects. For version 2 of the R package **ctsem**, we include a Bayesian specification and fitting routine that uses the **Stan** probabilistic programming language, via the **rstan** package in R. This allows for all parameters of the dynamic model to individually vary, using an estimated population mean and variance, and any time independent covariate effects, as a prior. Frequentist approaches with ctsem are documented in a JSS publication (Driver, Oud, and Voelkle 2017), and in R vignette form at `https://cran.r-project.org/package=ctsem/vignettes/ctsem.pdf`. The Bayesian approach is discussed more fully and conceptually in a preprint article (Driver and Voelkle 2016)available at `https://www.researchgate.net/publication/310747801_Hierarchical_Bayesian_Continuous_Time_Dynamic_Modeling`, but here we provide more specifics on the software for getting started with the software of the Bayesian approach.

Keywords: hierarchical time series, Bayesian, longitudinal, panel data, state space, structural equation, continuous time, stochastic differential equation, dynamic models, Kalman filter, R.

1. Overview

1.1. Subject Level Latent Dynamic model

This section describes the fundamental subject level model, and where appropriate, the name of the ctModel argument used to specify specific matrices. The description of the full model, including subject level likelihood and population model, is provided in Driver and Voelkle (2016), available at `https://www.researchgate.net/publication/310747801_Hierarchical_Bayesian_Continuous_Time_Dynamic_Modeling`. 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, although the meaning

is of course somewhat different.

1.2. Subject level latent dynamic model

The subject level dynamics are described by the following stochastic differential equation:

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

Vector $\boldsymbol{\eta}(t) \in \mathbb{R}^{\upsilon}$ represents the state of the latent processes at time t. The matrix $\mathrm{A} \in \mathbb{R}^{\upsilon \times \upsilon}$ (DRIFT) represents the so-called 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 $\mathrm{b} \in \mathbb{R}^{\upsilon}$ (CINT), in combination with 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 (or shape) of time dependent predictors. 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 (2017).

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

Here, time dependent predictors $\mathrm{x}_u \in \mathbb{R}^{l}$ (tdpreds) are observed at measurement occasions $u \in \mathrm{U}$. 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 x_u. The effect of these impulses on processes $\boldsymbol{\eta}(t)$ is then $\mathrm{M} \in \mathbb{R}^{\upsilon \times l}$ (TDPREDEFFECT).

$\mathrm{W}(t) \in \mathbb{R}^{\upsilon}$ (DIFFUSION) represents independent Wiener processes, with a Wiener process being a random-walk in continuous time. $\mathrm{dW}(t)$ 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 $\mathrm{G} \in \mathbb{R}^{\upsilon \times \upsilon}$ represents the effect of this noise on the change in $\boldsymbol{\eta}(t)$. Q, where $\mathrm{Q} = \mathrm{GG}^{\mathsf{T}}$, represents the variance-covariance matrix of the diffusion process in continuous time.

1.3. Subject level measurement model

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

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

$\mathrm{y}(t) \in \mathbb{R}^{c}$ is the vector of manifest variables, $\Lambda \in \mathbb{R}^{c \times \upsilon}$ (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 $\Theta \in \mathbb{R}^{c \times c}$ (MANIFESTVAR).

1.4. 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, then a range of matrix transformations are applied to generate the continuous time matrices described, as well as all relevant discrete time instantiations (More variability in measurement time intervals thus means more computations). The higher level distribution has a multivariate normal prior. A more comprehensive description is found at the end of this document.

1.5. Install software and prepare data

Install ctsem software:

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

Prepare data in long format, each row containing one time point of data for one subject. We need a subject id column containing numbers from 1 to total subjects, rising incrementally with each subject going down the data structure. This is to ensure coherence with the internal structure of the Stan model. The column is named by default "id", though this can be changed in the model specification. We also need a time column "time", containing 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
[1,]  1 -0.05 -0.776  5.26  1.6153 -1.08039 -1.607  0.0324
[2,]  1  1.25 -0.310  4.91  0.4244 -1.08039 -1.607  0.0324
[3,]  1  2.10 -1.167  5.14  0.6623 -1.08039 -1.607  0.0324
[4,]  1  3.03  0.336  5.16  0.0561 -1.08039 -1.607  0.0324
[5,]  1  3.98 -0.134  4.97  2.3243 -1.08039 -1.607  0.0324
[6,]  2 36.06 -5.006 -5.79  0.9433 -0.00619  0.643 -0.5037
[7,]  2 36.88 -6.287 -4.41 -0.0220 -0.00619  0.643 -0.5037
[8,]  2 37.87 -7.257 -4.00 -0.4195 -0.00619  0.643 -0.5037
```

At present, missingness is fine on manifest indicators, but not allowed elsewhere.

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 classic **ctsem** behaviour and prepares an **OpenMx** model. Other arguments to ctModel proceed as normal, although some matrices used for type 'omx' are not relevant for the Stan formats, either because the between subject matrices have been removed, or because time dependent and independent predictors are now treated as fixed regressors and only require effect (or design) matrices. These differences are documented in the help for ctModel.

Argument	**Sign**	**Default**	**Meaning**
n.manifest	c		Number of manifest indicators per individual at each measurement occasion.
n.latent	v		Number of latent processes.
LAMBDA	Λ		n.manifest × n.latent loading matrix relating latent to manifest variables.
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 × n.latent matrix of latent process initial covariance, specified with standard deviations on diagonal and (partial) correlations on lower triangle.
T0MEANS	η_1	free	n.latent × 1 matrix of latent process means at first time point, T0.
MANIFESTMEANS	τ	free	n.manifest × 1 matrix of manifest means.
MANIFESTVAR	Θ	free diag	lower triangular matrix of var / cov between manifests, specified with standard deviations on diagonal and (partial) correlations on lower triangle.
DRIFT	A	free	n.latent × n.latent matrix of continuous auto and cross effects.
CINT	b	0	n.latent × 1 matrix of continuous intercepts.
DIFFUSION	Q	free	lower triangular n.latent × n.latent matrix containing standard deviations of latent process on diagonal, and (partial) correlations 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 × 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.

```
model<-ctModel(type='stanct',
  n.latent=2, latentNames=c('eta1','eta2'),
  n.manifest=2, manifestNames=c('Y1','Y2'),
  n.TDpred=1, TDpredNames='TD1',
  n.TIpred=3, TIpredNames=c('TI1','TI2','TI3'),
  LAMBDA=diag(2))
```

This generates 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 the model, and three time independent predictors. Additional complexity or restrictions may be added, the table below shows the basic arguments one may consider and their link to the dynamic model parameters. For more details see the ctsem help files or papers. Note that for the Stan implementation, ctModel requires variance covariance matrices (DIFFUSION, T0VAR, MANIFESTVAR) to be specified with standard deviations on the diagonal, correlations (partial, if > 2 latent processes) the lower off diagonal, and zeroes on the upper off diagonal.

These matrices may all be specified using a combination of character strings to name free parameters, or numeric values to represent fixed parameters.

The parameters subobject of the created model object shows the parameter specification that will go into Stan, 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.

```
head(model$pars,8)
```

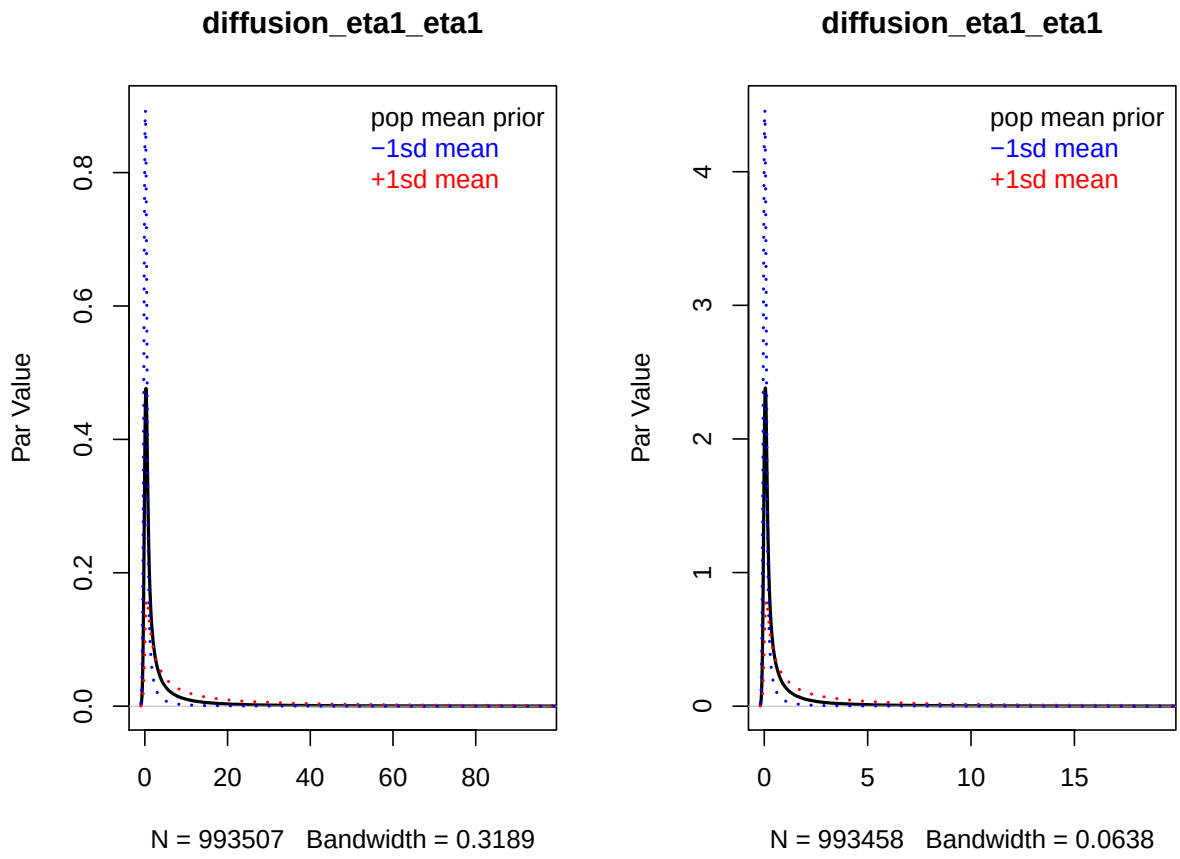
```
   matrix row col           param value                      transform indvarying sdscale
1 TOMEANS   1   1     TOmean_eta1    NA                    (param) * 10       TRUE       1
2 TOMEANS   2   1     TOmean_eta2    NA                    (param) * 10       TRUE       1
3  LAMBDA   1   1            <NA>     1                            <NA>      FALSE       1
4  LAMBDA   1   2            <NA>     0                            <NA>      FALSE       1
5  LAMBDA   2   1            <NA>     0                            <NA>      FALSE       1
6  LAMBDA   2   2            <NA>     1                            <NA>      FALSE       1
7   DRIFT   1   1 drift_eta1_eta1    NA -log(exp(-param*1.5)+1)-.00001       TRUE       1
8   DRIFT   1   2 drift_eta1_eta2    NA                         (param)       TRUE       1
  TI1_effect TI2_effect TI3_effect
1       TRUE       TRUE       TRUE
2       TRUE       TRUE       TRUE
3      FALSE      FALSE      FALSE
4      FALSE      FALSE      FALSE
5      FALSE      FALSE      FALSE
6      FALSE      FALSE      FALSE
7       TRUE       TRUE       TRUE
8       TRUE       TRUE       TRUE
```

One may modify the output model to either restrict between subject differences (set some parameters to not vary over individuals), alter the transformation used to determine the prior / bounds, or restrict which effects of time independent predictors to estimate. Plotting the original prior, making a change, and plotting the resulting prior, are shown here – in this case we believe the stochastic latent process innovation for our first latent process, captured by row 1 and column 1 of the DIFFUSION matrix, to be small, so scale our prior accordingly to both speed and improve sampling. Rather than simply scaling by 0.2 as shown here, one could also construct a new form of prior, so long as the resulting distribution was within the bounds required for the specific parameter. Note that the resulting distribution is a result of applying the specified transformation to a standard normal distribution, with mean of 0 and standard deviation of 1. To change the underlying standard normal, one would need to edit the resulting Stan code directly.

```
par(mfrow=c(1,2))
plot(model,rows=11,hypersd=1)
print(model$pars$transform[11])
```

```
[1] "exp(param*2)"
```

```
model$pars$transform[11]<- "(exp(param*2) +.0001)*.2"
plot(model, rows=11, hypersd=1)
```



The plots show the prior distribution for the population mean of DIFFUSION[1,1] in black, as well as two possible priors for the subject level parameters, conditional on our specified population (hyper) standard deviation of 1. The blue prior results from assuming the population mean is one standard deviation lower than the mean of the prior, and the red one standard deviation higher. This latter prior can be scaled using the sdscale column of the parameters subobject.

Restrict between subject effects as desired. Unnecessary between subject effects will slow sampling and hinder appropriate regularization, but 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. The prior scale for between subject variance may need to be restricted when limited data (in either the number of time points or number of subjects) is available, to ensure adequate regularisation. Here we restrict MANIFESTVAR effects between subjects, and set all prior scales for the standard deviation of the population distribution to 0.2, from the default of 1.0. A rough interpretation of this change in sdscale is simply that we expect lower values for the population standard deviation, but to better interpret the effect of this latter change, see the section on standard deviation transformations.

```
model$pars[c(15,18),]$indvarying<-FALSE
model$pars$sdscale[1:28] <- .5
```

Also restrict which parameters to include time independent predictor effects for in a similar way, for similar reasons. In this case, the only adverse effects of restriction are that the relationship between the predictor and variables will not be estimated, but the subject level

parameters themselves should not be very different, as they are still freely estimated. Note that such effects are only estimated for individually varying parameters anyway – so after the above change there is no need to set the tipredeffect to FALSE for the T0VAR variables, it is assumed. Instead, we restrict the tipredeffects on all parameters, and free them only for the manifest intercept parameters.

```
model$pars[,c('TI1_effect','TI2_effect','TI3_effect')]<-FALSE
model$pars[c(19,20),c('TI1_effect','TI2_effect','TI3_effect')]<-TRUE
```

1.7. Model fitting

Once model specification is complete, the model is fit to the data using the ctStanFit function as follows – depending on the data, model, and number of iterations requested, this can take anywhere from a few minutes to days. Current experience suggests 300 iterations is often enough to get an idea of what is going on, but more may be necessary for robust inference. This will of course vary massively depending on model and data. For the sake of speed for this example we only sample for 200 iterations with a lowered max_treedepth - this latter parameter controls the maximum number of steps the Hamiltonian sampler is allowed to take per iteration, with each increase of 1 doubling the maximum. With these settings the fit should take only a few minutes (but will not be adequate for inference!). Those that wish to try out the functions without waiting, can simply use the already existing ctstantestfit object instead of creating the fit object (and adjust the code in following sections as needed!).

```
fit<-ctStanFit(datalong = ctstantestdat, ctstanmodel = model, iter=200,
  chains=2, plot=FALSE, control=list(max_treedepth = 6))
```

```
In file included from C:/Users/driver/Documents/R/win-library/3.4/BH/include/boost/config.hpp:39:0,
                 from C:/Users/driver/Documents/R/win-library/3.4/BH/include/boost/math/tools/confi
                 from C:/Users/driver/Documents/R/win-library/3.4/StanHeaders/include/stan/math/rev
                 from C:/Users/driver/Documents/R/win-library/3.4/StanHeaders/include/stan/math/rev
                 from C:/Users/driver/Documents/R/win-library/3.4/StanHeaders/include/stan/math/rev
                 from C:/Users/driver/Documents/R/win-library/3.4/StanHeaders/include/stan/math/rev
                 from C:/Users/driver/Documents/R/win-library/3.4/StanHeaders/include/stan/math.hpp
                 from C:/Users/driver/Documents/R/win-library/3.4/StanHeaders/include/src/stan/mode
                 from file180c322373c7.cpp:8:
C:/Users/driver/Documents/R/win-library/3.4/BH/include/boost/config/compiler/gcc.hpp:186:0: warning
 #  define BOOST_NO_CXX11_RVALUE_REFERENCES
 ^
<command-line>:0:0: note: this is the location of the previous definition
```

The plot argument allows for plotting of sampling chains in real time, which is useful for slow models to ensure that sampling is proceeding in a functional manner. Models with many parameters (e.g., many subjects and all parameters varying over subject) may be too taxing for the function to handle smoothly - we have had succcess with up to around 4000 parameters.

1.8. Summary

After fitting, the summary function may be used on the fit object, which returns details

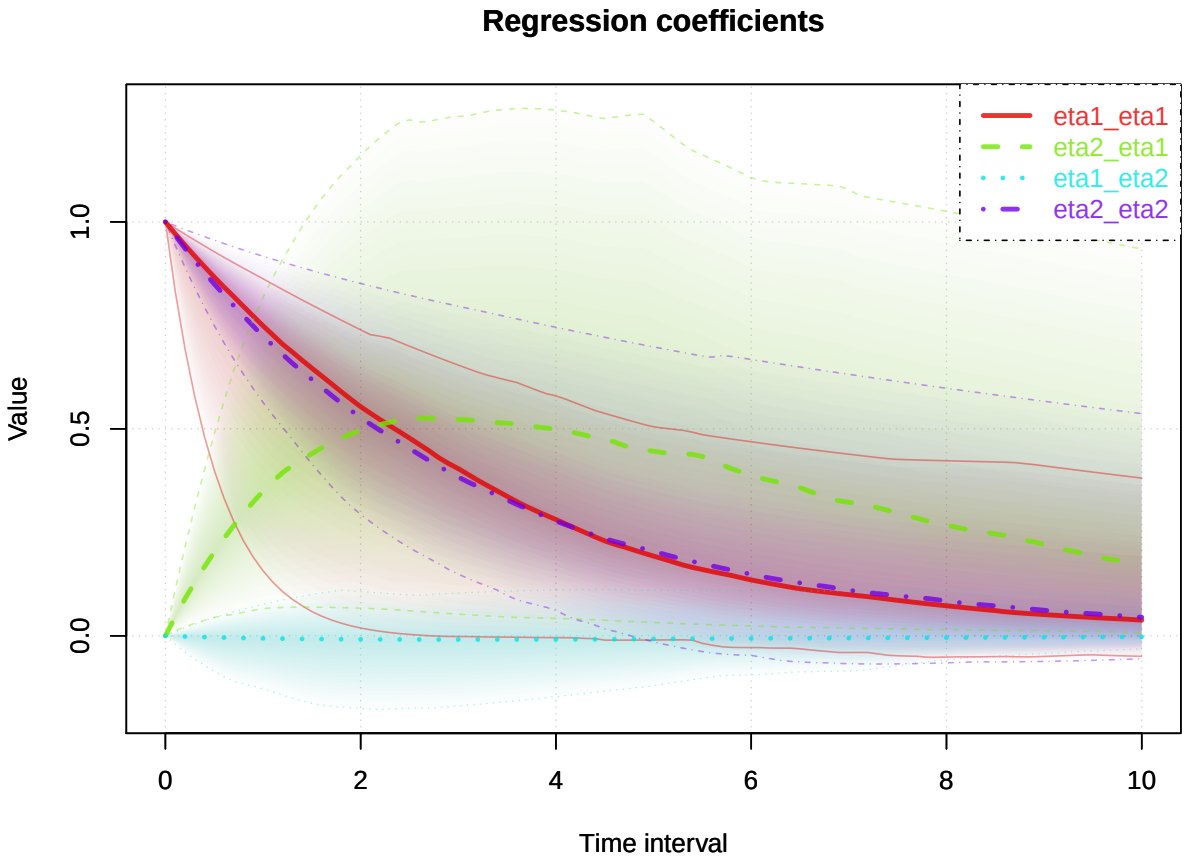
regarding the population mean parameters, population standard deviation parameters, population correlations, and the effect parameters of time independent predictors. Additionally, summary outputs a range of matrices regarding correlations betweeen subject level parameters. hypercorr_means reports the posterior mean of the correlation between raw (not yet transformed from the standard normal scale) parameters. hypercorr_sd reports the standard deviation of these parameters. hypercovcor_transformedmean reports the correlation between transformed parameters on the lower triangle, the variance of these parameters on the diagonal, and the covariance on the upper triangle. To view the posterior median of the continuous time parameter matrices, the ctStanContinuousPars function can be used.

```
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') are by default from a time interval of 1, but this can be changed.

1.9. Plotting

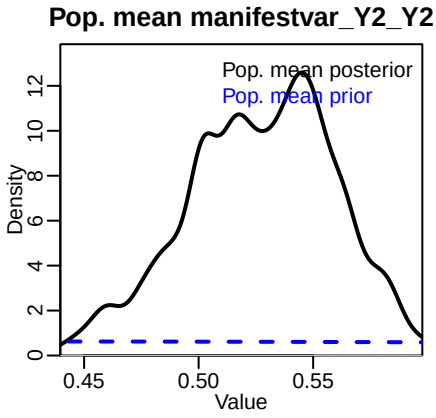
The plot function outputs a sequence of plots, all generated by specific functions. The name of the specific function appears in a message in the R console, checking the help for each specific function and running them separately will allow more customization. Some of the plots, such as the trace, density, and interval, are generated by the relevant rstan function and hopefully self explanatory. The plots specific to the heirarchical continuous time dynamic model are as follows:



The above plot 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 of these coefficients.

The relation between posteriors and priors for variables of interest can also be plotted as follows:

```
ctStanPlotPost(obj = fit, rows=11)
```



Shown are approximate density plots based on the post-warmup samples drawn. For each parameter four plots are shown – the first displays the posterior distribution of subject level parameters, the subject level prior (generated from repeated sampling of the hyper parame-

ters), and the prior for the population mean.

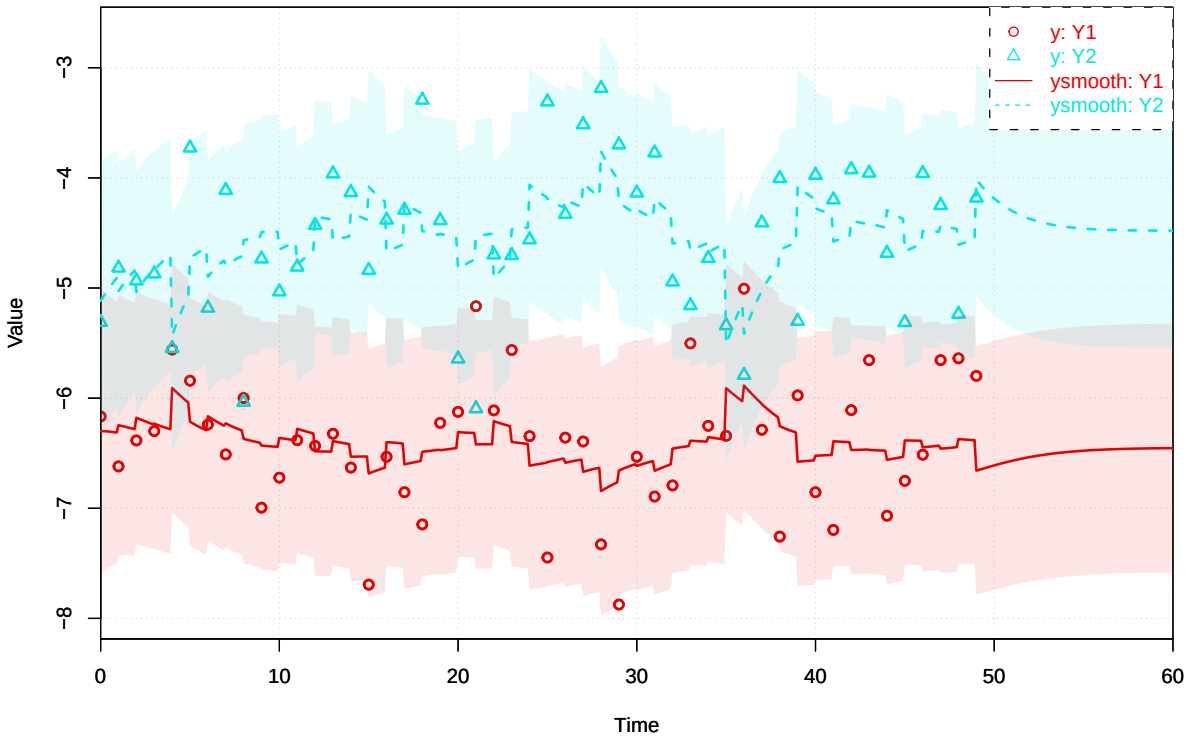
1.10. Stationarity

When it is reasonable to assume that the prior for long term expectation and variance of the latent states is the same as (or very similar to) the prior for initial expectations and variances, setting some form of stationarity in advance may be beneficial. Three approaches to this are possible. The first approach is to name free parameters of the T0VAR or T0MEANS matrix `'stationary'`, which can be useful if for instance only the initial variance should be stationary, or just one of the initial variances. Another approach is to set the argument stationary=TRUE to ctStanFit. Specifying this argument then ignores any T0VAR and T0MEANS matrices in the input, instead replacing them with asymptotic expectations based on the DRIFT, DIFFUSION, and CINT matrices. Alternatively, a prior can be placed on the stationarity of the dynamic models, calculated as the difference between the T0MEANS and the long run asymptotes of the expected value of the process, as well as the difference between the diagonals of the T0VAR covariance matrix and the long run asymptotes of the covariance of the processes. Such a prior encourages a minimisation of these differences, and can help to ensure that sensible, non-explosive models are estimated, and also help the sampler get past difficult regions of relative flatness in the parameter space due to colinearities between the within and between subject parameters. However if such a prior is too strong it can also induce difficult dependencies in model parameters, and there are a range of models where one may not wish to have such a prior. To place such a prior, the model$stationarymeanprior and model$stationaryvarprior slots can be changed from the default of NA to a numeric vector, representing the normal standard deviation of the deviations from stationarity. The number of elements in the vector correspond to the number of latent processes.

1.11. Individual level analyses

Individual level results can also be considered, as ctsem includes functionality to output prior (based on all prior observations), updated (based on all prior and current observations), and smoothed (based on all observations) expectations and covariances from the Kalman filter, based on specific subjects models. For ease of comparison, expected manifest indicator scores conditional on prior, updated and smoothed states are also included. This approach allows for: predictions regarding individuals states at any point in time, given any values on the time dependent predictors (external inputs such as interventions or events); residual analysis to check for unmodeled dependencies in the data; or simply as a means of visualization, for comprehension and model sanity checking purposes. An example of such is depicted below, where we see observed and estimated 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).

```
ctKalman(fit, subjects=2, timerange=c(0,60), kalmanvec=c('y', 'ysmooth'),
  timestep=.1, plot=TRUE)
```



1.12. 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. 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.

1.13. Using Rstan functions

The standard rstan output functions such as summary and extract are also available, and the shinystan package provides an excellent browser based interface. The stan fit object is stored under the $stanfit subobject from the ctStanFit output. The parameters which are likely to be of most interest in the output are prefixed by "hmean_" for hyper (population) mean, "hsd_" for hyper standard deviation, and "tipred_" for time independent predictor. Any hmean parameters are returned in the form used for input - so correlations and standard deviations for any of the covariance related parameters. Subject specific parameters are denoted by the matrix they are from, then the first index represents the subject id, followed by standard matrix notation. For example, the 2nd row and 1st column of the DRIFT matrix for subject 8 is "$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. The exception to this are the time independent predictor effects, prefixed with "*tipred_*", for which a linear effect of a change of 1 on the predictor is approximated. So although "tipred_TI1" is only truly linear with respect to internal parameterisations, we approximate the linear effect by averaging the effect of a score of +1 or -1 on the predictor, on the population mean. For

any subject that substantially differs from the mean, or simply when precise absolute values of the effects are required (as opposed to general directions), they will need to be calculated manually.

1.14. Oscillating, single subject example - sunspots data

In the following example we fit 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 – some differences are due to the contrast between Bayes and maximum likelihood, though if desired one could adjust the code to fit using maximum likelihood, as here we have only one subject.

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

#setup model
 model <- ctModel(type='stanct', n.latent=2, n.manifest=1,
  manifestNames='sunspots',
  latentNames=c('ss_level', 'ss_velocity'),
   LAMBDA=matrix(c( 1, 'ma1' ), nrow=1, ncol=2),
   DRIFT=matrix(c(0, 'a21', 1, 'a22'), nrow=2, ncol=2),
   MANIFESTMEANS=matrix(c('m1'), nrow=1, ncol=1),
   CINT=matrix(c(0, 0), nrow=2, ncol=1),
   TOVAR=matrix(c(0,0,0,1), nrow=2, ncol=2), #Because single subject
   DIFFUSION=matrix(c(0, 0, 0, "diffusion"), ncol=2, nrow=2))

 model$pars$indvarying<-FALSE #Because single subject
 model$pars$transform[14]<- '(param)*5+44 ' #Because not mean centered
 model$pars$transform[4]<-'log(exp(-param*1.5)+1)' #To avoid multi modality

#fit
fit <- ctStanFit(datalong, model, iter=300, chains=2)

In file included from C:/Users/driver/Documents/R/win-library/3.4/BH/include/boost/config.hpp:39:0,
                 from C:/Users/driver/Documents/R/win-library/3.4/BH/include/boost/math/tools/confi
                 from C:/Users/driver/Documents/R/win-library/3.4/StanHeaders/include/stan/math/rev
                 from C:/Users/driver/Documents/R/win-library/3.4/StanHeaders/include/stan/math/rev
                 from C:/Users/driver/Documents/R/win-library/3.4/StanHeaders/include/stan/math/rev
                 from C:/Users/driver/Documents/R/win-library/3.4/StanHeaders/include/stan/math/rev
                 from C:/Users/driver/Documents/R/win-library/3.4/StanHeaders/include/stan/math.hpp
                 from C:/Users/driver/Documents/R/win-library/3.4/StanHeaders/include/src/stan/mode
                 from file180c35bb1ce0.cpp:8:
C:/Users/driver/Documents/R/win-library/3.4/BH/include/boost/config/compiler/gcc.hpp:186:0: warning
 #  define BOOST_NO_CXX11_RVALUE_REFERENCES
 ^
<command-line>:0:0: note: this is the location of the previous definition

#output
summary(fit)$popmeans
```

```
                                    mean    sd   2.5%    50%  97.5% n_eff  Rhat
hmean_T0mean_ss_level              24.122 6.148 10.298 24.728 34.545 119.5 1.019
hmean_T0mean_ss_velocity           14.138 7.512 -0.449 13.991 27.612 175.6 1.004
hmean_ma1                           0.659 0.204  0.257  0.653  1.094 112.0 0.997
hmean_a21                          -0.354 0.043 -0.442 -0.353 -0.269 138.9 1.005
hmean_a22                          -0.323 0.081 -0.492 -0.317 -0.183 108.1 1.009
hmean_diffusion                    15.450 2.414 11.463 15.069 20.826  97.2 1.003
hmean_manifestvar_sunspots_sunspots  1.487 1.222  0.127  1.122  4.515 119.1 1.008
hmean_m1                           46.369 2.765 40.928 46.536 51.030 264.9 1.008
```

1.15. Population standard deviations - understanding the transforms

Internally, we sample parameters that we will refer to here as the 'raw' parameters – these parameters have no bounds and are drawn from normal distributions. Both population mean (internally: hypermeans) and subject level (internally: indparamsbase) raw parameters are drawn from a normal(0, 1) distribution. Depending on the specific parameter, various transformations may be applied to set appropriate bounds and priors. The population standard deviation (hypersd) for these raw parameters is sampled (by default) from a normal(0, 1) distribution called rawhypersd, which is by default transformed via an exponential function – ensuring standard deviation parameters are positive and have a prior distribution that could be considered a regularised independence Jeffreys prior. This distribution can 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 50 subjects. Although we sample the priors here, this is merely to reflect the prior and enable understanding and plotting.

```
#population mean and subject level deviations (pre-transformation)

hypermeans_prior <- rnorm(99999, 0, 1)
hypermeans_post <- -2 #hypothetical sample

indparamsbase_prior <- rnorm(99999, 0, 1)
indparamsbase_post <- rnorm(50, 0, 1) #hypothetical sample

#population standard deviation prior

rawhypersd_prior <- rnorm(99999, 0, 1)
hypersd_prior <- exp(rawhypersd_prior * 2 )

#population standard deviation posterior

hypersd_post <- .4 #hypothetical

#square root of population correlation matrix
hypercorrchol_post <- 1 #because only 1 parameter here...

#population cholesky covariance matrix
#here based on mean of hypersd_post, for convenience...
#in reality would have multiple samples.
hypercovchol <- diag(hypercorrchol_post,1) %*%
```

```
  diag(hypersd_post,1) %*% diag(hypercorrchol_post,1)

#subject level parameters
#first compute pre transformation parameters
#then transform appropriately (here according to drift auto effect)
indparams <- hypercovchol %*% indparamsbase_post + hypermeans_post
indparams <- -log(exp(-1.5 * indparams) + 1)

#post transformation population standard deviation
hsd_ourparameter <- abs( #via delta approximation
  (-log(exp(-1.5 * (hypermeans_post + hypersd_post)) + 1) -
   -log(exp(-1.5 * (hypermeans_post - hypersd_post)) + 1) ) / 2)
```

2. The model

For a comprehensive description of the statistical model, see http://www.researchgate.net/publication/310747801_Hierarchical_Bayesian_Continuous_Time_Dynamic_Modeling.

References

Driver CC, Oud JHL, Voelkle MC (2017). "Continuous Time Structural Equation Modeling with R Package Ctsem." *Journal of Statistical Software*, **77**(5). ISSN 1548-7660. doi:10.18637/jss.v077.i05.

Driver CC, Voelkle MC (2016). "Hierarchical Bayesian Continuous Time Dynamic Modeling." *Manuscript submitted for publication.*

Driver CC, Voelkle MC (2017). "Understanding the Time Course of Interventions with Continuous Time Dynamic Models." *Manuscript submitted for publication.*

Tómasson H (2013). "Some Computational Aspects of Gaussian CARMA Modelling." *Statistics and Computing*, **25**(2), 375–387. ISSN 0960-3174, 1573-1375. doi:10.1007/s11222-013-9438-9.

Affiliation:

Charles Driver
Center for Lifespan Psychology
Max Planck Institute for Human Development
Lentzeallee 94, 14195 Berlin
Telephone: +49 30 82406-367 E-mail: driver@mpib-berlin.mpg.de
URL: http://www.mpib-berlin.mpg.de/en/staff/charles-driver