

A Tutorial in Dynamic Systems Simulation and Modelling With R, ctsem, and lme4: Couples' Affect Dynamics Over Time

Charles C. Driver

Michael Aristodemou

Understanding how people change is a fundamental goal of psychological science. However, translating complex ideas about psychological dynamics into formal models can be challenging without the right tools. In this tutorial, we introduce a workflow that leverages R and the ctsem package to help researchers build and understand dynamic systems models that capture the complexity of psychological processes. Our workflow emphasizes iterative model-building through simulations, model fitting, and visualizations of the model-implied dynamics alongside their fit to data. We begin with familiar linear models in lme4 and gradually transition to ctsem, which allows us to incorporate complexities such as state-dependent change, random fluctuation distinct from measurement error, covariate effects, interactions, and external inputs. These modelling concepts are illustrated using a running example of affect dynamics in couples therapy, demonstrating how key conceptual and methodological ideas in dynamic systems modelling come together. Our aim is to provide a general framework for understanding dynamic systems modelling and to encourage further exploration of theory-driven statistical approaches in psychological research.

Introduction

Dynamic systems theory provides a framework for understanding how psychological constructs evolve over time ([van Geert 2011](#)). To turn conceptual ideas about dynamic systems into insight about mechanisms of change, theory must be translated into empirically testable models, which when combined with data can in turn suggest updates to theory([Borsboom et al. 2021](#)). Yet, this translation from conceptual ideas to testable statistical models can be difficult without tools that make the steps transparent.

This tutorial is designed to help psychological researchers bridge the gap between theoretical concepts in dynamic systems and their practical implementation using statistical modelling.

We begin with familiar linear mixed-effects models using the R package lme4 ([Bates et al. 2015](#)), then gradually move into the dynamic systems and continuous-time domain using the package ctsem ([Driver and Voelkle 2018a](#)). This progression highlights how modelling choices can evolve alongside theory, allowing increasingly realistic and nuanced representations of psychological change.

A key strength of ctsem is that it can accommodate this full modelling continuum. Starting from basic linear growth models, one can move through discrete- and continuous-time versions of classic frameworks such as the Cross-Lagged Panel Model ([Kenny 2005](#)), the random intercept extension of it ([Hamaker, Kuiper, and Grasman 2015](#)), and related forms such as network vector autoregression ([Epskamp 2020](#)). From there, one can extend ctsem to models with considerably more complexity. These extensions include complex variation across individuals, oscillations, and measurement or dynamics parameters that shift over time or in response to moderators and interventions. In this way, ctsem provides a flexible environment for building models up step by step: researchers can begin with simple forms to establish a baseline, then incorporate the dynamics that theory suggests, then perhaps go further with data-driven model and theory development.

Continuous-time representations are particularly valuable for theory-oriented model development, because of the more natural link between continuous-time parameters and typical theoretical interests ([Aalen et al. 2016](#); [Ryan and Hamaker 2022](#); [Driver and Tomasik 2023](#)). Because most psychological processes – such as emotion regulation, stress responses, or cognitive fluctuations – are thought to evolve continuously rather than only when measured, continuous-time models provide a natural representation. Parameters such as drift (state dependence), diffusion (system noise), and external inputs can be aligned much more closely with mechanistic interpretations, instead of as artifacts of the measurement schedule. This interpretability makes continuous-time models especially valuable for theory-oriented research, as it ensures that any constraints imposed on the model (e.g., fixing the effect of one variable on another to zero) are meaningfully interpretable with respect to theory ([Driver 2025](#)).

To make these ideas concrete, we present a running example: the impact of couples therapy on affect dynamics over time. This example is grounded in empirical research showing that partners' affective states are dynamically interdependent, becoming synchronized through processes of mutual influence ([Butler and Randall 2013](#)). Such coupling can either amplify or dampen emotional experiences, with implications for individual well-being and relationship stability. Using this context, we progressively build from basic assumptions of linear change to models incorporating feedback loops, random fluctuations, dyadic interdependence, and the effects of therapeutic inputs. Through this step-by-step approach, we introduce different models of affective processes that reflect increasing levels of complexity:

1. **Steady (Linear) Change:** an affect process that changes at a steady rate from an initial state of affect during each observation.
2. **Nonlinear Change (Discrete Time):** an affect process whose growth rate depends on a person's level of affect at the observation before the change happens—creating a

nonlinear growth process.

3. **Nonlinear Change with Random Fluctuations (Continuous Time)**: an affect process that changes continuously over time, and whose rate of change depends on a person's level of affect at the moment of change—creating a nonlinear growth process. Here we include randomness in the process, reflecting unpredictable events that generate fluctuations in affect. The conceptual shift necessary to understand modelling dynamics in continuous time is explained prior to the introduction of this model.
4. **Couple Dynamics with Self and Partner Effects (Continuous Time)**: moving to multivariate models, an affect process for a couple whose rate of change is dependent on their own level of affect and their partner's level of affect at a given moment.
5. **Couple Dynamics with Individual Differences and Time-Independent Moderation (Continuous Time)**: in this model, we introduce individual differences in system dynamics and a time-independent moderator (i.e. partner effects are moderated by the overall time they spend together).
6. **Couple Dynamics with Individual Differences and Time-Dependent Moderation (Continuous Time)**: in this model, the partner effects are moderated by a time-varying moderator tracking whether the couple is together (time-dependent covariate).
7. **Nonlinear Change with a Transient External Shock (Continuous Time)**: here the affect process is impacted by a therapy session that only directly impacts affect during the session.
8. **Nonlinear Change with a Persistent External Shock (Continuous Time)**: here the affect process is impacted by a therapy session and the therapy session induces persistent changes in the process.

By going through the steps to formalize these eight processes, researchers will have a structured approach to modelling affect (or other psychological constructs) dynamics in a way that captures both within-person change and between-person heterogeneity.

To end this tutorial, we showcase a process of iterative model evaluation and improvement starting from a misspecified model and working toward the data generating mechanism.

Steady (Linear) Change in Affect

Let us start by considering a hypothetical patient named Jill. Jill started going to couples therapy and has agreed to track her affect once every seven days (i.e. at equidistant time intervals), for 21 weeks, through a mobile app. What we want, is to conceptualize possible ways that Jill's affect develops during this time. So we develop some ideas about what Jill's development might look like, and pick the simplest one to start. Our first idea, is that Jill's affect will increase at a steady rate throughout the course of couples therapy.

Now we want to turn our conceptual idea of linear change into a statistical model so that we can fit it to data and obtain estimates of parameters. For a steady increase in affect, a linear

regression model is an appropriate choice. With our linear regression model, we state that Jill's observed affect (y) changes at a steady rate (B) as a function of time (t) from an initial affect state (η_{t0}), along with deviations from this processes that reflect imperfect measurements and or (we will distinguish these in later sections) an imperfect model:

$$y(t) = \eta_{t0} + Bt + \epsilon(t)$$

At any time point t , Jill's observed affect equals her starting level (η_{t0}) plus a constant amount of growth per unit time (Bt), with some random measurement error ($\epsilon(t)$). The parameter B represents how much affect increases each week (with random deviations from this due to measurement error).

Next, we can obtain a better understanding of the growth process it implies (and whether it matches our conceptual idea) by generating data from the model (along with specific parameter values) and visualizing the resulting trajectory of affect.

Simulation

So let's write some code to generate data from a linear regression model representing a steady rate of change in affect and visualize its model-implied trajectory. The first code chunk below specifies a plotting function that we will re-use throughout the tutorial – details on this are beyond the scope of this tutorial, but essentially it contains a set of instructions for how to plot the data using the `ggplot2` package (Wickham 2011), you should run this code before proceeding with further code chunks.

```
# Define reusable plotting functions
plot_trajectory <- function(data, x_var = "Time", y_var = "Affect",
                           x_label = "Time (weeks)", y_label = "Affect",
                           color_var = NULL, show_legend = FALSE,
                           y2_var = NULL) {
  p <- ggplot(data, aes_string(x = x_var))

  # Handle single variable or couple plots
  if (is.null(y2_var)) {
    # Single variable plot
    p <- p + aes_string(y = y_var)
    if (!is.null(color_var)) {
      p <- p + aes_string(color = color_var)
    }
    p <- p + geom_point() + geom_line()
  } else {
    # Couple plot with two variables
    p <- p +
      geom_line(aes_string(y = y_var)) +
      geom_line(aes_string(y = y2_var), linetype = "dashed") +
      geom_point(aes_string(y = y_var))
  }

  p <- p + theme_bw() + labs(x = x_label, y = y_label)
}
```

```

if (!is.null(color_var) && is.null(y2_var)) {
  p <- p + labs(color = color_var)
}

if (!show_legend) {
  p <- p + theme(legend.position = "none")
}

return(p)
}

```

Moving to the actual simulation code, we first specify the number of time points we wish to generate data for (`Time <- 0:20`). Second, we set the initial state of affect at the beginning of the growth process, i.e. the beginning of therapy (`initialAffect <- 5`). Third, we specify the linear regression model that will formalize our linear affect process for a single person, whose affect starts at an initial level (`initialAffect`) of 5 and increases steadily by 0.51 during every weekly observation (`Time*.51`). Fourth, we add random noise to our model by sampling values from a normal distribution with a mean of 0 and a standard deviation of 1 (`rnorm(n=length(Time), mean = 0, sd = 1)`). These noise values are added to each affect value that is generated by our model. Fifth, we create a data frame to store the number of time points (i.e. weeks) and the affect values we generated for each time point. Finally, we use the created function `plot_trajectory` to plot the data using a lineplot to illustrate the model-implied affect process over the course of 21 weeks of therapy.

```

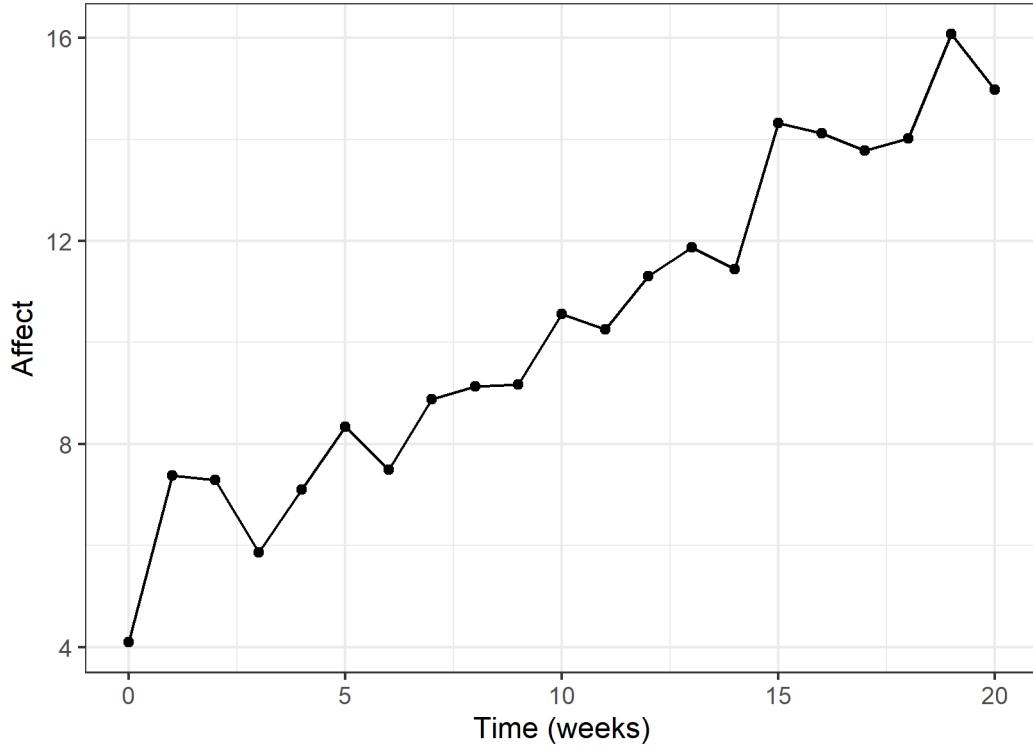
Time <- 0:20 # Generate time points for each measurement occasion
initialAffect <- 5 # Initial level of mood at week 0

# Specify a linear model that generates affect data increases by 0.51 each week
Affect <- initialAffect + Time*.51 + rnorm(n=length(Time), mean = 0, sd = 1)

# Create a data frame to store the generated data
data <- data.frame(Time = Time, Affect = Affect)

# Plot the data using a lineplot
plot_trajectory(data)

```



Here we see our linear model provides a good representation of the idea that Jill's affect increases at a steady pace each time we observe her. Albeit with some random fluctuations, which we can attribute to measurement error.

Dealing with Individual Differences in Affect Changes

By viewing Jill's model-implied growth process it strikes us that we have no idea whether her rate of change is slow or fast relative to other people who go to couples therapy. That is we may want to know if couples therapy leads to a different amount of change for different people, and if differences in people's rate of change are related to their initial level of affect. For instance, people who have a much lower initial level of affect than Jill may benefit more from couples therapy. To understand where Jill sits relative to other patients, we need to introduce individual differences into our model.

The equation for a linear model which allows people to vary in their initial affect level and rate of change, looks very similar to the linear regression model presented above but includes an i subscript to indicate that each person has their own initial level (η_{t0i}) and rate of change ($B_i t$) of affect:

$$y_i(t) = \eta_{t0i} + B_i t + \epsilon_i(t)$$

The subscript i indicates that each person i has their own parameters. Person i 's affect starts at their own initial level (η_{t0i}) and grows at their own rate (B_i). For example, Person 1 might start at affect level 3 and grow by 0.7 units per week, while Person 2 might start at affect level 6 and grow by 0.4 units per week. This allows us to model both within-person change and between-person differences.

This model is commonly termed a linear mixed-effects model (Bates et al. 2015), because it can include both fixed effects (parameters that are assumed to be the same across all subjects, often interpreted as population-level effects) and random effects (parameters that allow for individual variations, such as subject-specific intercepts η_{t0i} or slopes $B_i t$).

Simulation

We now generate data from our linear mixed-effects model. The code below is used to generate data for 20 subjects (`NSubjects <- 20`) whose affect is measured 21 times, once per week for 21 weeks (`times <- seq(from=0, to=20, by=1)`). To generate a different initial state for each subject, we sample 20 initial states from a normal distribution of initial affect levels with a mean of 5 and a standard deviation of 2 (`initialAffect <- rnorm(n = NSubjects, mean = 5, sd = 2)`). The values here are arbitrary and can be tweaked to adjust the standard deviation of initial affect (`sd`), and the population average initial affect level (`mean`). To match our hypothetical scenario, where people's rate of growth in affect is related to their initial level, we can generate data so that people's rate of change is negatively correlated with people's initial level of affect (`timeCoefficients <- rnorm(n = NSubjects, mean = 0.5, sd = 0.1) + scale(initialAffect) * -0.2`). In this calculation we scale (z-score) the initial affect level, ensuring it always has a mean of 0 and a standard deviation of 1, which can make it easier to think about the meaning of the coefficient (-0.2 in this case). We then check the correlation between the initial affect level and the rate of change in affect to ensure it is as expected.

```
# Generate data for multiple subjects with individual differences
NSubjects <- 20
times <- seq(from=0, to=20, by=1) # sequence of time points when subjects are measured
Nobs <- length(times) # number of observations per subject
# sample initial affect for each subject from a normal distribution
initialAffect <- rnorm(n = NSubjects, mean = 5, sd = 2)
# sample rate of change in affect for each subject
timeCoefficients <- rnorm(n = NSubjects, mean = 0.5, sd = 0.1) +
  scale(initialAffect) * -0.2 # include negative influence of initial affect
cor(initialAffect, timeCoefficients) # check correlation in individual differences
```

```
[,1]
[1,] -0.9314609
```

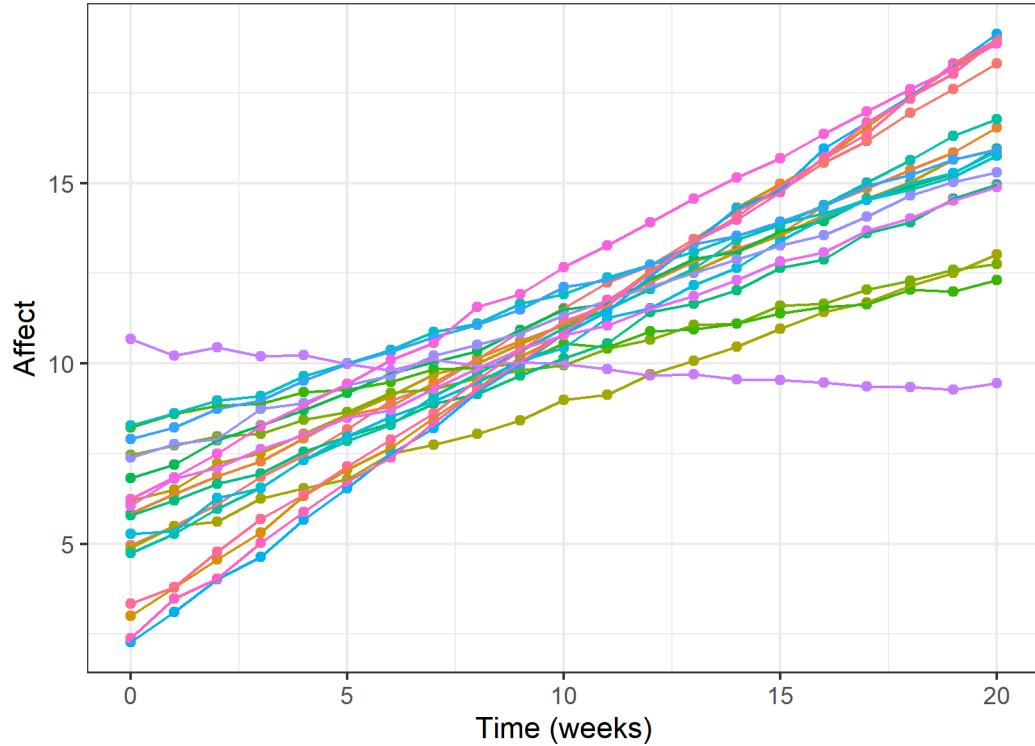
We can now create an empty data frame that can later hold the data we generate for each subject and each observation. To fill the data frame, we create two nested loops. The first loop (`for(subi in 1:NSubjects)`) allows us to go through each subject (`subi`) one by one. For each subject selected by the first loop, we initialize the second loop (`for(obsi in 1:Nobs)`) that allows us to go through each observation (`obsi`). Within this loop, we fill in each row in our data frame, which corresponds to an observation. For each observation, we generate information about a person's affect level, the current time point (week), and their subject identifier. To keep track of the current observation for each iteration of our loop, we initialize a row count at 0 (`row <- 0`) and then add 1 to this running row count (`row <- row + 1`). To reflect the imperfect measurement of affect, we add some random noise to the affect data that we generated, by sampling random values from a normal distribution with a mean of 0 and a standard deviation of 1 (`data$Affect <- data$Affect + rnorm(n=nrow(data), mean = 0, sd = .1)`).

```
#create empty data.frame to fill step by step
data <- data.frame(Subject= rep(NA,NSubjects*Nobs),
Time = rep(NA,NSubjects*Nobs),
Affect = rep(NA,NSubjects*Nobs))

row <- 0 #initialize row counter, to track which row of the data.frame we are on
for(subi in 1:NSubjects){
  for(obsi in 1:Nobs){ #for each observation of a subject
    row <- row + 1 # add an integer to the row to keep track of the current observation (row)
    data$Affect[row] <- initialAffect[subi] + # currentt affect is initial affect plus...
      times[obsi] * timeCoefficients[subi] # the effect of time that has passed since.
    data$Time[row] <- times[obsi] # store time point for current observation (row)
    data$Subject[row] <- subi # store subject identifier for current observation (row)
  }
}

# add random noise to the affect data
data$Affect <- data$Affect + rnorm(n=nrow(data), mean = 0, sd = .1)

# plot subject specific trajectories, color-coded by subject
plot_trajectory(data, color_var = "as.factor(Subject)")
```



From our plot, we can see that the correlated initial state and rate of change capture the idea that individuals who start with lower affect levels may improve faster over time.

Model Fitting

We can now fit our model to the data we generated and interpret how well it captures the patterns in our observations. To specify and fit our linear mixed-effects models, we will use the `lme4` package in R (explicitly including intercepts, although they would be included by default):

```
# Fit linear mixed-effects model
library(lme4)
lme_model <- lmer(Affect ~ # predict Affect
  1 + # a fixed effect for the intercept
  Time + # with a fixed effect of time
  (1 + Time | Subject), # random intercept and random effect (slope) of time per subject
  data = data) # specify data to fit model to
```

After fitting the model to our data, we can ask for a summary of its parameter estimates.

```

# Summary of model output
summary(lme_model)

Linear mixed model fit by REML ['lmerMod']
Formula: Affect ~ 1 + Time + (1 + Time | Subject)
Data: data

REML criterion at convergence: -422

Scaled residuals:
    Min      1Q  Median      3Q     Max 
-3.04945 -0.62547  0.02896  0.67221  2.41418 

Random effects:
 Groups   Name        Variance Std.Dev. Corr
 Subject (Intercept) 4.48950  2.1188
          Time         0.05055  0.2248  -0.93
 Residual           0.01024  0.1012
Number of obs: 420, groups: Subject, 20

Fixed effects:
            Estimate Std. Error t value
(Intercept) 5.86278   0.47388 12.37
Time         0.50327   0.05028 10.01

Correlation of Fixed Effects:
  (Intr) 
Time -0.933

```

For this simple affect process, we can get a grasp on the model-implied trajectory through the numerical estimates offered by lme4's output table. We can see that our linear mixed-effects model, recovers the parameter values from the data generating model well. We can find the population-level estimates for the intercept and slope values under "Fixed effects". The variance of the subject-specific deviations from the population average can be found under "Random effects". The correlation between the individual differences in the intercept and slope values is also under the "Random Effects" section (do not be misled by the "Correlation of Fixed Effects" heading, it does not reflect individual difference correlations). For more information about lme4 and its functionalities we refer the reader to the package documentation ([Bates et al. 2015](https://cran.r-project.org/web/packages/lme4/lme4.pdf)) <https://cran.r-project.org/web/packages/lme4/lme4.pdf>.

Visualizing Predictions

Once our models become more complicated, a useful tool to understand their predictions is visualization. Visualization is a great way to build intuition about the process our models imply and allows us to detect sources of model misfit that may be difficult to identify if solely relying on numerical fit indices (see Gabry et al. (2019) for examples). For our lme4 model, we can generate two plots showing: (1) the model's predictions (solid line) and its associated uncertainty (shaded ribbon) based only on the estimated model parameters for a single subject (subject 3); (2) predictions based on the model parameters and all data, (past, present, and

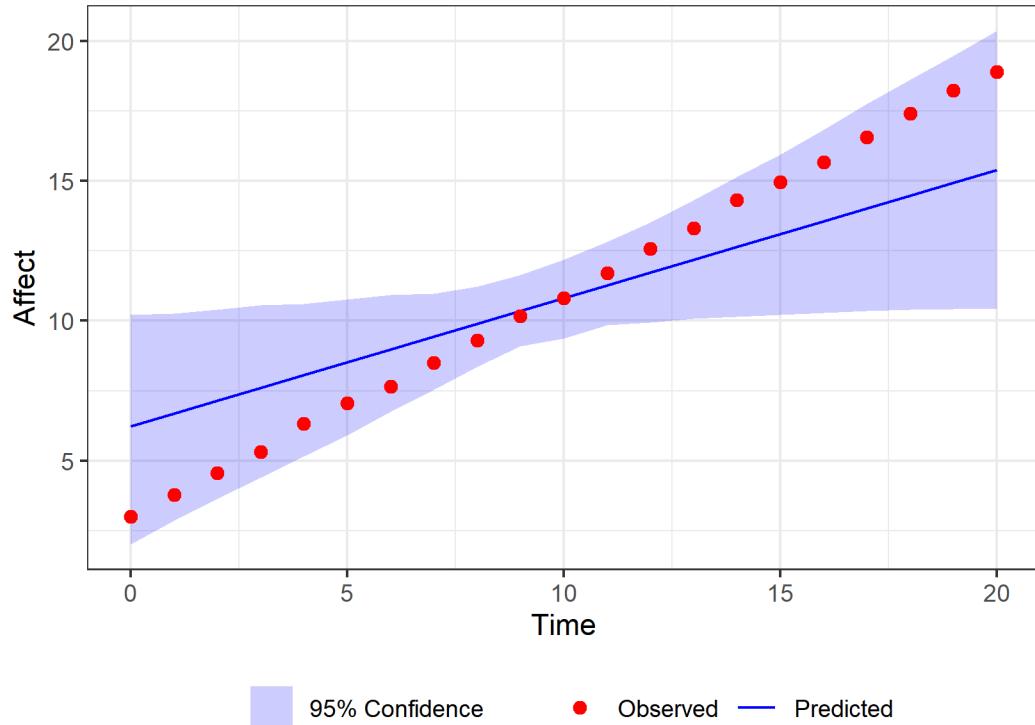
future) of a single subject (subject 3). The code below uses the `bootMer` function to generate bootstrap (i.e., many) predictions from the model, and then uses the `ggplot2` package to visualize the predictions and uncertainty.

```
# Define a function to extract predictions
pred_fun <- function(model) {
  predict(model, data)
}

# Use bootMer to generate bootstrap (i.e., many) predictions
boot_preds <- bootMer(lme_model, FUN = pred_fun, nsim = 100)

# Extract the predicted values and calculate the mean and confidence intervals
newdata <- data.frame(data, #create a new data frame including original data
  pred = apply(boot_preds$t, 2, mean),
  lower = apply(boot_preds$t, 2, quantile, 0.025),
  upper = apply(boot_preds$t, 2, quantile, 0.975))

# Use ggplot2 to visualize the predictions and uncertainty
ggplot(newdata[newdata$Subject==3,], aes(x = Time)) +
  geom_line(aes(y = pred, color = "Predicted")) +
  geom_ribbon(aes(ymin = lower, ymax = upper, fill = "95% Confidence"), alpha = 0.2) +
  geom_point(aes(y = Affect, color = "Observed"), size = 2) +
  scale_color_manual(name = "", values = c("Predicted" = "blue", "Observed" = "red")) +
  scale_fill_manual(name = "", values = c("95% Confidence" = "blue")) +
  labs(y = "Affect", x = "Time") +
  theme_bw() + theme(legend.position = "bottom")
```

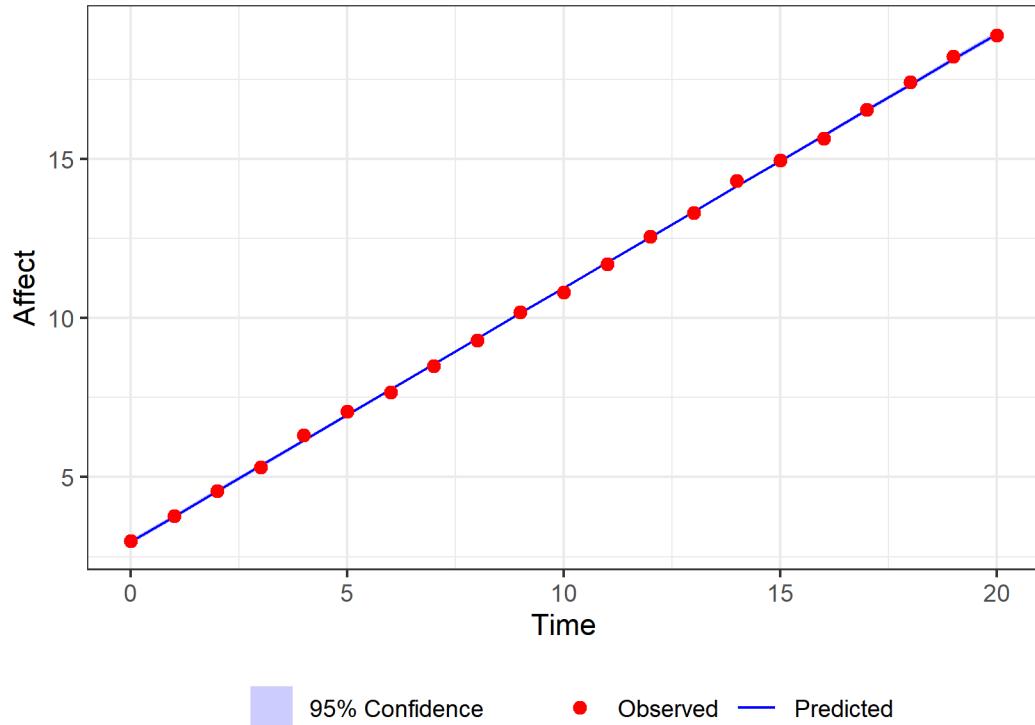


The first plot shows how well information solely from the estimated model parameters can reproduce our pattern of observations. The red dots show the observed affect values at each time point. The solid blue line shows the model implied expectation of affect, with the shaded blue area surrounding the line showing the variation / uncertainty about this trajectory. We can see that by solely relying on the model expectation, our model does a relatively poor job of predicting the actual data points. The solid line does not map well onto the observed trajectory of affect, and there is a large amount of uncertainty about the model-implied trajectory.

```
# Define a function to extract predictions
preds <- predict(lme_model, se.fit = TRUE)

# create a new data frame including original data and predictions
newdata <- data.frame(data, preds,
  lower = preds$fit - 1.96 * preds$se.fit,
  upper = preds$fit + 1.96 * preds$se.fit)

# Use ggplot2 to visualize the predictions and uncertainty
ggplot(newdata[newdata$Subject==3], #just visualise for subject 3
  aes(x = Time, y = fit)) +
  geom_line(aes(color = "Predicted")) +
  geom_ribbon(aes(ymin = lower, ymax = upper, fill = "95% Confidence"), alpha = 0.2) +
  geom_point(aes(y = Affect, color = "Observed"), size = 2) +
  scale_color_manual(name = "", values = c("Predicted" = "blue", "Observed" = "red")) +
  scale_fill_manual(name = "", values = c("95% Confidence" = "blue")) +
  labs(y = "Affect", x = "Time") +
  theme_bw() + theme(legend.position = "bottom")
```



In the second plot, we visualize the model's predictions after it has learned from all the observations in our dataset. So, data from past, present, and future observations are used by the model to predict each observation. Here the model-implied trajectory does an excellent job at mapping onto our observations – having learnt the individual difference parameters needed for this subject specifically, predictions are far more accurate.

Nonlinear Change (Discrete Time)

By formalizing our linear growth process for Jill and other patients, we observe that our simple conceptual idea leads to a surprising prediction. Namely, therapy actually helps people with a lower initial level of affect to have a higher affect level at the end of the 21 weeks of therapy, compared to people who started with higher affect. This means that if Jill entered therapy with more severe affect problems she would come out of therapy with a higher level of affect than if she entered with less severe affect problems. Moreover, if we take our model with a steady rate of growth seriously, then this would imply that Jill could endlessly increase her affect levels by continuing therapy. Both of these predictions seem unrealistic. So we have to rethink our model of affect (or interpret results from it **very cautiously**).

A more defensible prediction could be to expect that people's rate of growth slows down as their affect improves. This would mean that people with a low initial level of affect would increase faster at the start, relative to people with a higher initial level of affect, but their growth rate would slow down as their affect improves. In a statistical model, we can implement such a (nonlinear) growth process by allowing each person's slope to vary not just as a function of where they start, but also as a function of where they are *at every point in time*.

Our statistical model of this nonlinear affect process can still be represented using a regression model. We can achieve our nonlinear growth model by computing the observed level of affect at each time point ($\eta(t)$) as a function of affect at the previous time point ($A\eta_{t-1}$), plus a constant value (B) that is added to each observation:

$$\eta(t) = A\eta_{t-1} + B$$

At each time point t , affect depends on two components: (1) a fraction A of the previous affect level, and (2) a constant input B . The parameter A controls state dependence – if A is positive but less than 1 (e.g., $A = 0.8$), affect carries forward from the previous time point but at a reduced rate, creating a dampening effect – with only this state-dependency fraction, no matter the initial value of affect (whether positive or negative), the process would always converge towards zero. Including the constant input B allows the process to converge towards a different, non-zero value (which is not exactly B , but a function of both A and B). In case of an A between 0 and 1, and a B value greater than zero, the process will converge towards a positive value – whether the process starts above this value or below. Essentially, this creates nonlinear growth: when affect is low, the total of the state-dependency and constant

input is larger (faster growth). As affect increases, growth slows down, as the loss from the state-dependency fraction becomes similar to the gain from the constant input B .

With this equation we can emulate a process where the rate of growth in affect dampens throughout the course of a therapy session. Our code will do the heavy lifting of computation and allow us to visualize the model-implied trajectory that our equation creates. But we can also build some intuition by solving this equation for a small number of time points.

So, let's consider 3 time points T_0 , T_1 , and T_2 . Then let's set the autoregressive coefficient (A) at 0.8, which means that 0.8 of the previous observation will be added to the current observation. Let's also add a constant of 2, as our continuous intercept (B) which is added to every observation.

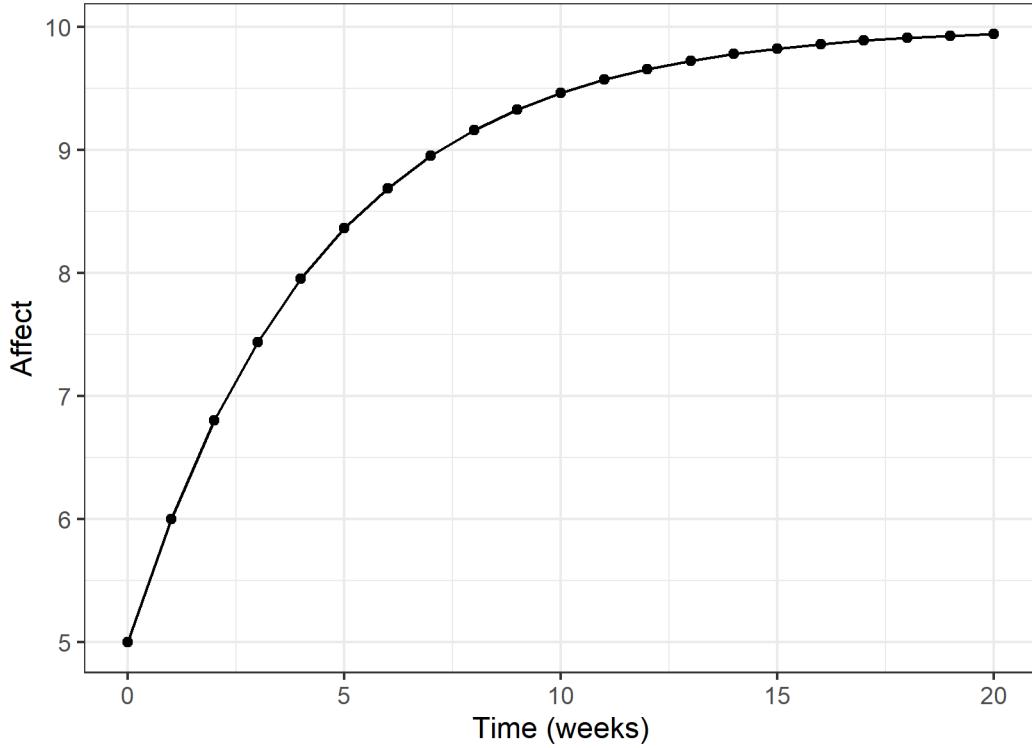
Then, we can start up our process at T_0 by setting an initial level of affect to 5 ($\eta(0) = 5$). At T_1 , we can see that $0.8 * 5 + 2$ leads to a current level of affect $\eta(1) = 6$. By applying our equation to the next time point, T_2 , we can see that $0.8 * 6 + 2$ leads to current level of affect equal to 6.8. So from T_0 to T_1 our affect level grew by 1 unit, but from T_1 to T_2 it only grew by 0.8. Thus, the rate of growth is dampening as consequence of a change in the prior level of affect.

Simulation

Below we provide code that does this computation for all of our 21 weeks, using the equation we just discussed. First we need to set the degree to which the current level of affect is dependent on the prior level of affect (i.e. state dependence, $A <- .8$). Then we add a constant value which is added to each observation generating a consistent rate of growth ($B <- 2$). By adding the state dependence term with the continuous intercept, we generate a nonlinear growth trajectory. We then set an initial affect level which is used to initialize our process (`initialAffect <- 5`). An if-else statement is used so that the first observation sets the initial level of affect (`if(i==1) Affect[i] <- initialAffect`) and each subsequent observation has an affect value that is generated by our model equation (`else Affect[i] <- A*Affect[i-1] + B`).

```
times <- seq(from=0, to=20, by=1) #generate sequence of time points when subjects are measured
A <- .8 #autoregressive coefficient / state dependence
initialAffect <- 5
B <- 2 #continuous intercept
Affect <- rep(NA,length(times)) #create empty affect vector
for(i in 1:length(times)){ #for each measurement occasion
  if(i==1) Affect[i] <- initialAffect #if first time point, set to initial affect
  else Affect[i] <- A*Affect[i-1] + B #otherwise, use autoregression and continuous intercept
}

# Plot the data using a lineplot
plot_trajectory(data.frame(Affect=Affect, Time=Time))
```



Our figure shows a nonlinear increase in affect over time. This is a better representation of a reality where therapy cannot make you infinitely happy, but it may help you recover from a low point.

A Conceptual Shift to Continuously Evolving Processes

Our model now provides a more realistic depiction of Jill's growth in affect from week to week. But what happens in between each of our weekly observations? One possibility is that affect stays flat and suddenly jumps during each of our weekly observations. That is, change only happens when we are looking. If we try to imagine such a discretely changing affect process, it would look like a staircase of affect. The alternative possibility is that people's affect changes continuously across time, forming a coherent trajectory. The rate of change that we observe from week to week, is then an aggregate of all the moment-to-moment changes that happen between our weekly observations. If the second option is more akin to our expectation of reality, then a continuous-time framework can better represent the process. This becomes particularly important when modelling multiple interacting processes or when dealing with data collected at varying time intervals (For details on such issues we refer the reader to [Driver 2025](#); [Voelkle et al. 2012](#); [Kuiper and Ryan 2018](#)).

Differential Equations – The Mathematics of Continuous Time

To formally represent continuously changing processes we need to use differential equations. But even without a technical understanding of differential equations researchers can form an intuitive understanding of continuous processes and apply them to their own substantive domain. Our intuition can be aided by building a mental image of a continuously changing process. To do this we can imagine a discretely changing process (i.e., jumping from one time point to the next) where the size of the steps we take in time are extremely small.

Modelling a growth process in continuous time will be no different than in discrete time, in that we will use symbols to represent different factors and their relationships which combine to form our growth process. To understand the equation that generates our hypothesized affect process we can again visualize the model-implied process using simulations.

For example, we can model an affect process that changes at a steady rate in continuous time, akin to our first example where we used a linear regression model – but this time using a differential equation. The major difference here is that our outcome variable (left-hand side of equation) is the *rate of change*, rather than the *level*, of affect – level was the outcome in our discrete time (regression) representation. This rate of change in affect (η) at a given moment in time t is denoted by the derivative $\frac{d\eta}{dt}$. We can think of the rate of change at a given moment in time as what we would get if we were to glance at a speedometer in our car. The velocity that we would see, would tell us how fast our current position is changing at a particular moment in time. Since our model is linear, the rate of change at a given moment is given by a constant value B – there is no variation in the rate of change (or slope), it remains constant over time. Thus, the differential equation for a linear model is:

$$\frac{d\eta}{dt} = B$$

The notation $\frac{d\eta}{dt}$ reads as “the rate of change of η with respect to time t ” – think of it as affect’s “velocity” at any instant. If $B = 0.51$, then at every moment, affect is increasing at a constant rate of 0.51 units per week. This is like a car traveling at a constant speed, always travelling 100 km/h. Similarly, no matter Jill’s current affect level, it’s always increasing by 0.51 units per week.

If we want to pick up where we left off and represent our nonlinear growth model in continuous time, we can add a term that allows the rate of growth at a given time point to depend on the current state of affect (i.e. a state dependency):

$$\frac{d\eta}{dt} = A\eta(t) + B$$

Now the rate of change (velocity) depends on where affect currently is, as well as on the constant input B . The term $A\eta(t)$ means the rate of change is proportional to current affect level. If

A is negative (e.g., $A = -0.2$), higher affect leads to slower growth (or even decline), creating a self-regulating system – just as we saw in our discrete time model when the autoregression coefficient A was positive but less than 1. Here in the continuous-time case A is still a regression coefficient, but the outcome variable is now the **rate of change in affect**, rather than the **level of affect at a new time point**. The term B is still a constant upward push.

Simulation

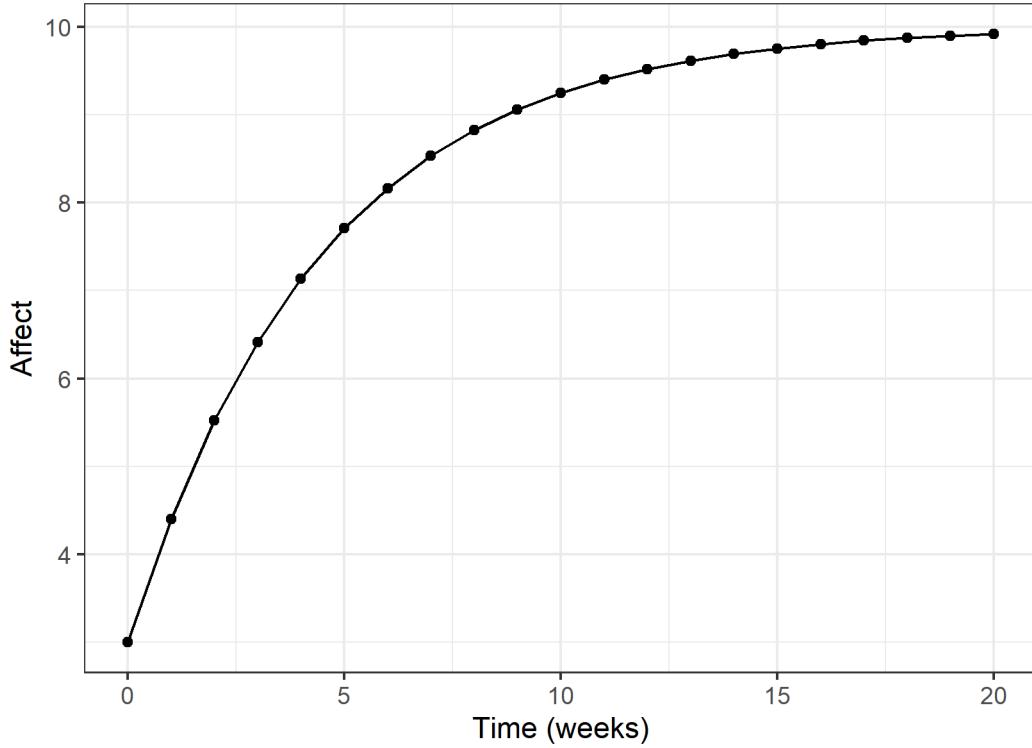
To generate data for this continuous growth process in R, we can approximate the solution to the differential equation using a ‘Euler-Maruyama’ method ([Higham 2001](#)) (or technically in this case without system noise, we would call this just the ‘Euler method’). This method works by breaking continuous time into small time slices (steps) and updating the state of our growth process at each step based on the model’s dynamics.

To generate data, we first need to define values for our model parameters. This includes: (1) the degree to which the current level of affect influences its own rate of change (A , continuous time state-dependence). (2) A constant input to the growth process (B , continuous intercept), and an initial level of affect (`initialAffect`). Then we create an empty vector to store the affect values given by our for-loop which implements the ‘Euler-Maruyama’ method (`Affect <- rep(NA,Nobs)`). The for-loop sets the initial level of affect for the first observation (`i=1`) to initialize the process (`if(i==1) Affect[i] <- initialAffect`). For each subsequent observation ($i > 1$), we compute the current level of affect by summing the *rate of change* at the prior time point (`dAffect = A*Affect[i-1] + B`) with level of affect at the prior time point (`Affect[i-1]`). That is, `Affect[i] <- Affect[i-1] + dAffect * 1`. To add the size of the desired time step into the equation, we multiply the rate of change by the size of the time step, in this case `dAffect` is multiplied by 1, meaning that the rate of change is equivalent to the rate of change between each observation.

```
times <- seq(from=0, to=20, by=1) # generate sequence of time points when subjects are measured
Nobs <- length(times) # number of observations per subject
A <- -.2 # continuous time state dependence
initialAffect <- 3 # set initial level of affect
B <- 2 # continuous intercept
Affect <- rep(NA,Nobs) # create empty affect vector

for(i in 1:Nobs){ # for each time point
  if(i==1) Affect[i] <- initialAffect # if first time point, set to initial affect
  else{
    dAffect <- A*Affect[i-1] + B # compute slope of affect at earlier time point
    Affect[i] <- Affect[i-1] + dAffect * 1 # update affect using slope and time step
  }
}

# Lineplot of the approximate continuous trajectory of affect
plot_trajectory(data.frame(Affect=Affect, Time=Time))
```



This is a somewhat crude approximation however – it assumes the rate of change is constant over each time interval of 1, when in fact our model specifies that the rate of change varies whenever affect changes, which is continuously. To better approximate a continuous time solution, we can take smaller steps in time. But more steps means that the necessary computation will be more intensive. So let's start with a small number of intermediary steps, say 10 steps in the time between our observations, to better approximate the continuous time process `Nsteps <- 10`. Now, we introduce another loop, within our previous for-loop, which splits the interval between each observation (e.g. week) into 10 smaller steps (`istep`). The first line within this loop represents the differential equation which gives us the rate of change at the prior time point (`dAffect = A*AffectState + B`). The line below it, updates the current level of affect based on the estimated rate of change at the prior time point multiplied by $1/10$ (i.e. $1/Nsteps$). The effect of this multiplication is to give us the rate of change over the smaller step of time we chose. In essence, this scales the rate of change from the full unit of time (e.g., week), given by the preceding equation, to one-tenth, allowing us to better approximate the underlying continuous growth process.

```
times <- seq(from=0, to=20, by=1) #generate sequence of time points when subjects are measured
Nobs <- length(times) #number of observations per subject
A <- -.2 #continuous time state dependence
B <- .5 #continuous intercept
initialAffect <- 3
B <- 2 #continuous intercept
```

```

Affect <- rep(NA,Nobs) #create empty affect vector
Nsteps <- 10 #number of steps in time to compute between each observation (increased precision)

for(i in 1:Nobs){ #for each time point
  if(i==1) Affect[i] <- initialAffect #if first time point, set to initial affect
  else{ #compute new affect state by taking a sequence of small steps in time
    AffectState <- Affect[i-1] #initialise with state at previous time point
    for(stepi in 1:Nsteps){ #take Nsteps in time between each observation
      dAffect <- A*AffectState + B #compute slope of affect at earlier time point
      AffectState <- AffectState + dAffect * 1/Nsteps #update state using slope and time step
    }
    Affect[i] <- AffectState
  }
}

#plot_trajectory(data.frame(Affect=Affect, Time=Time))

```

Nonlinear Change with Random Fluctuations (Continuous Time)

By using our previous model to predict Jill's affect trajectory, we would assume that her affect would change in a smooth deterministic manner. However, real-world affect processes are often subject to unpredictable changes that lead a person's affect to fluctuate in meaningful and persistent ways – perhaps Jill wakes up with a headache one day, or has a particularly nice conversation with a friend. These may not alter her long term trajectory, but can induce an effect both immediately and over some length of time. Such fluctuations are distinct from variation we might attribute to *measurement error* where we typically assume that imperfections in our measurements are random and independent at each occasion of measurement.

We can model random fluctuations in the affect process by adding a system noise term to our differential equation. With our current equation, the rate of change in affect $\frac{d\eta}{dt}$ is purely deterministic, meaning that how fast affect changes at a given moment is entirely determined by the terms in the right-hand side of the differential equation ($A\eta(t) + B$).

To model the impact of random influences on Jill's affect trajectory, we want to add a term that introduces a random “kick” to the rate of change at each moment. Meaning that aside from the deterministic effects there will be some random ups and downs in Jill's growth rate. To add a “kick” to Jill's rate of change we can introduce a noise term that is sampled from a normal distribution. This turns our (ordinary) differential equation into a stochastic differential equation ([Kloeden and Platen 1992](#)), with stochastic denoting the random noise component of our continuous process.

That is, we model our continuous time process as evolving over infinitesimal (read extremely small) steps in time. At each infinitesimal step in time, our noise term adds a portion of randomness to the rate of change. The amount that it adds is scaled by the system noise term G , defining the volatility of the process. Thus, our rate of change in affect $\frac{d\eta(t)}{dt}$ is a combination of some deterministic effect ($A\eta(t) + B$) and some random effect ($GdW(t)$), over each infinitesimal interval dt . Because of the stochastic nature of the random noise term, it

is not possible to analytically define its rate of change at any point in time. Thus, stochastic differential equations are often written a bit differently, with the dt term moved to the right-hand side of the equation, only applied to the deterministic portion:

$$d\eta = (A\eta(t) + B)dt + GdW(t)$$

This equation now has two components to change: (1) *Deterministic change* ($A\eta(t) + B)dt$: the predictable trend based on current affect level, and (2) *Random fluctuations* $GdW(t)$: unpredictable “shocks” that can push affect up or down randomly at any moment. This reflects the idea that while there is some predictability to change in affect over time, there are also unpredictable changes that can occur at any moment, which can persist over time

Simulation

Adding system noise. To add system noise to our continuous time model, we will update the data generating code using the Euler method discussed in the previous section. The system noise term will be added to the affect level that is computed for each of our chosen time steps (`istep`). Thus, when we update a person’s current affect state at each small time step we add some noise. This noise is sampled from a normal distribution with a mean of 0 and a standard deviation of 1 divided by `Nsteps` (this time 100 steps). Before being added to the function that generates the current affect state, each draw of random noise is multiplied by `G` (the system noise coefficient) to scale the amount of system noise. Thus, the higher the `G` value the more noisy the affect process becomes.

Individual differences. To generate data for multiple individuals we have to make some further tweaks to our code. First, for each participant we sample their initial affect level from a normal distribution with mean 5 and standard deviation of 2 (`initialAffect <- rnorm(n = NSubjects, mean = 5, sd = 2)`). Second, we add an empty data frame which includes a greater number of rows to fill. Third, we add a loop to repeat the computation of the affect process for each subject (`for(subi in 1:NSubjects)`).

```
# Generate data for multiple subjects with individual differences
NSubjects <- 20
times <- seq(from=0, to=20, by=1) #generate sequence of time points when subjects are measured
Nobs <- length(times) #number of observations per subject
initialAffect <- rnorm(n = NSubjects, mean = 5, sd = 2)
A <- -.1 #continuous time state dependence
B <- 1 #continuous intercept
G <- 0.2 #system noise coefficient

#create empty data.frame to fill step by step
data <- data.frame(Subject= rep(NA,NSubjects*Nobs),
                    Time = rep(NA,NSubjects*Nobs),
                    Affect = rep(NA,NSubjects*Nobs))

Nsteps <- 100 #number of steps in time to compute between each observation (more precise)
```

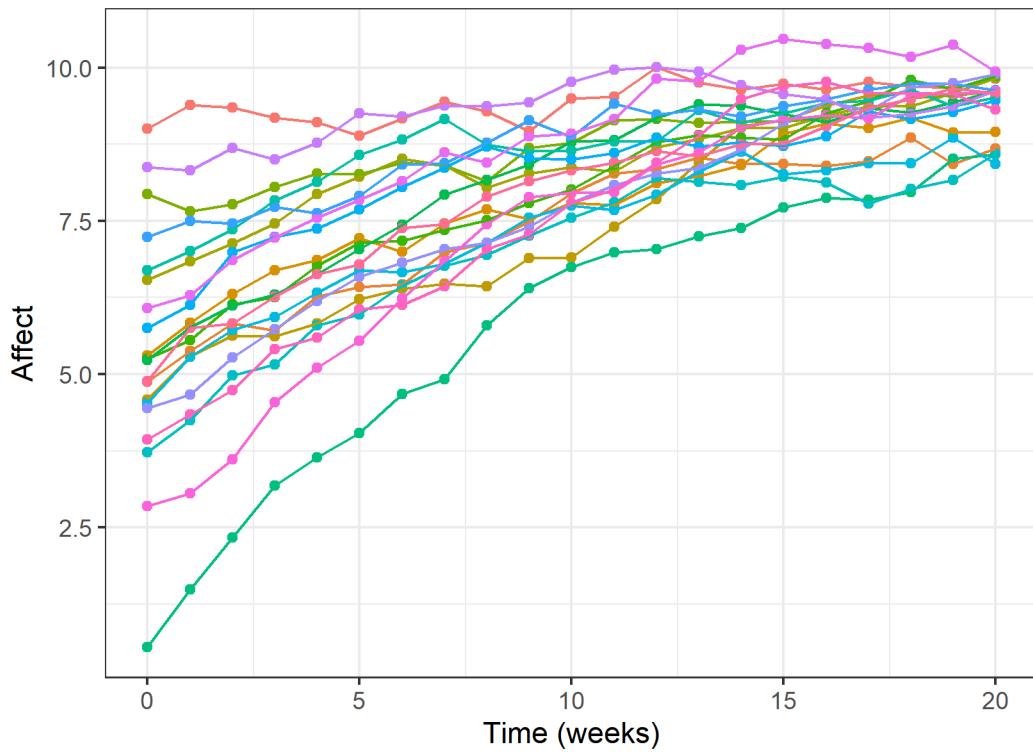
```

row <- 0 #initialize row counter, to track which row of the data.frame we are on
for(subi in 1:NSubjects){
  for(obsi in 1:Nobs){ #for each observation of a subject
    row <- row + 1
    if(obsi==1) AffectState <- initialAffect[subi] #if first time point, set to initial affect
    if(obsi>1){ #else compute new affect state by taking a sequence of small steps in time
      for(stepi in 1:Nsteps){ #take Nsteps in time between each observation
        dAffect <- A*AffectState + B #compute deterministic slope of affect at earlier time point
        AffectState <- AffectState + dAffect * 1/Nsteps + #update state using slope and time step
          G * rnorm(n=1, mean=0, sd=sqrt(1/Nsteps)) #and add system noise
      }
    }
    data$Affect[row] <- AffectState #input affect data
    data$Time[row] <- times[obsi] #input time data
    data$Subject[row] <- subi #input subject data
  }
}

data$Affect <- data$Affect + rnorm(n=nrow(data), mean = 0, sd = .05) #add measurement error

plot_trajectory(data, color_var = "as.factor(Subject)")

```



Now we have a somewhat more realistic depiction of affect dynamics, with fluctuations around the smooth, state-dependent increase in affect over time. This model captures the idea that affect processes are subject to both predictable trends and random fluctuations, reflecting some

of the complexity of real-world affect dynamics. Note also, that here individuals vary in their initial levels of affect, and the rate of change varies accordingly, such that therapy does not just help everyone at the same rate, and nor does it lead to unrealistic long term projections as we saw in the linear case – there is a limit.

ctsem: A Flexible Tool for Dynamic Systems Modelling

To fit a continuous-time model with state-dependence and system noise to data we make the shift to a ‘state-space’ framework (Durbin and Koopman 2012; Harvey 1989; Kalman 1960), allowing a measurement model to be linked to a latent dynamic model. The R package `ctsem` (Driver, Oud, and Voelkle 2017; Driver and Voelkle 2018a) provides a software framework for specifying and fitting continuous-time (CT) and discrete-time (DT) dynamic models. It enables researchers to model the behavior and evolution of processes over time, and its combination of features can offer some unique advantages for analyzing longitudinal data.

Software Installation

Before starting, ensure your system is ready to use `ctsem`. Follow these steps:

1. **Install R and RStudio:** Download and install R (<https://cran.r-project.org/>) and RStudio (<https://posit.co/products/open-source/rstudio/>) or alternative interfaces.
2. **Configure c++ for model compilation with Stan:** `ctsem` relies on Stan (Carpenter et al. 2017) for log likelihood calculations. Most of the models we will fit do not require compilation, but some of the more complex models we will fit do. Ensure you have a compatible C++ compiler installed:
 - **Windows:** Install RTools (<https://cran.r-project.org/bin/windows/Rtools/>).
 - **Mac:** Install Xcode command line tools by running `xcode-select --install` in the terminal.
 - **Linux:** Install `g++` and other build tools using your package manager (e.g., `sudo apt install build-essential`).
3. **Install Required Packages:**

```
install.packages(c("ctsem", "ggplot2", "lme4", "tinytex"))
```

Model Fitting

To fit the data we just generated with `ctsem`, we first need to specify our model using the `ctModel` function. The code below is annotated for convenience, for more information about the specification of a continuous-time model using the `ctModel` function please refer to the `ctsem` manual <https://cran.r-project.org/web/packages/ctsem/ctsem.pdf> (Driver, Voelkle, and Oud 2025). Here we will provide a quick description of some of the links between the model equation and the software specification.

Our `ctsem` model can be thought of as linking a latent (unobserved) ‘true’ affect process, to observed (or ‘manifest’) variables, which are part of our data. Observed variables are treated as indicators of the underlying latent process, and are linked to the latent states via a measurement model.

In this case our latent dynamic model is given by the stochastic differential equation:

$$d\eta = (A\eta(t) + B)dt + GdW(t)$$

This model is quite general, and can be formulated as very large matrices (when e.g., there are many latent processes interacting with each other and/or many observed variables). As such `ctsem` expects matrices as inputs for each of the different model components, but also contains some shortcuts to make it easier to specify the model. Model components can be either fixed to specific values (input a numeric value) or estimated from the data (input a string with the name of the parameter to estimate). When we move on to matrices later, each matrix can be a mixture of fixed and estimated values. Here is how we specify our current dynamic model using the `ctsem` syntax:

1. The `DRIFT` argument specifies the state dependence A , and contains a freely estimated parameter we call `stateDependence`, making the rate of growth depend on the current level of affect.
2. The `CINT` argument specifies the continuous intercept (B). We call the continuous intercept parameter `B` and only estimate a fixed effect (one value for all subjects). To switch off random effects (on by default for initial values and intercept parameters) we need to add `FALSE` into the 3rd ‘slot’ of the parameter, done by adding vertical bars after the parameter name, i.e. `B||FALSE`. We do not need to modify the 2nd slot, so it is left empty.
3. The `DIFFUSION` argument specifies the system noise term (G), and contains a freely estimated parameter `systemNoise`, allowing the model to capture random fluctuations around the model-implied growth trajectory.
4. The initial value of our latent process is treated as a random variable with its own mean and variance parameters: $\eta_{t0} = \mu_{0i} + \epsilon_{0i}$, where $\epsilon_{0i} \sim \mathcal{N}(0, \sigma^2)$. The `TOMEANS` argument, is used to specify the mean parameter for the initial level of affect, which we call `initialAffect`. By adding `TRUE` behind `initialAffect||` we are allowing for

random effects (between-subject variance) around the population estimate of the initial level of affect in our sample – this would have been enabled by default in any case.

The latent process model is linked to our observations using a (linear) measurement model:

$$\text{Affect}(t) = \Lambda\eta(t) + \tau + \epsilon(t)$$

Where $\text{Affect}(t)$ is the observed level of affect. Λ is a matrix of regression weights used to link the latent process to our observations, η is the latent level of affect, τ reflects a constant that can be used to shift the relationship between observations and a latent variable, and ϵ represents measurement error, and is distributed as a multivariate normal distribution with mean 0 and (usually diagonal) covariance matrix Θ .

1. The **LAMBDA** argument specifies the Λ matrix. In our case it is a 1x1 matrix with the value 1, meaning that the latent Affect (η) is directly and equally reflected in the observed Affect without any scaling.
2. The **MANIFESTMEANS** argument specifies the measurement intercept (τ) which is simply a constant that is added to the measurement model to shift the relationship between observations and a latent variable. In this case it is set to zero, implying that there is no systematic offset in the measurement process.
3. The **MANIFESTVAR** argument specifies the measurement error variance in terms of standard deviations (ϵ), which we call **residualSD**.

```
# Fit continuous time structural equation model
ct_model <- ctModel( #define the ctsem model
  # Specify features of the data
  manifestNames = "Affect", #names of observed variables in dataset
  latentNames = "Affect", #names of latent processes
  time = 'Time', #name of time column in dataset
  id = 'Subject', #name of subject column in dataset
  type='ct', #use continuous time / differential equation model (dt for discrete-time)
  # Specify features of the model
  MANIFESTVAR = 'residualSD', #sd of the residual / measurement error
  LAMBDA = matrix(1,nrow=1,ncol=1), #relating latent process to observed variables
  MANIFESTMEANS=0, #no measurement intercept (1 observed variable relates directly to latent)
  CINT='B|FALSE', #continuous intercept with *no* random effects
  TOMEANS='initialAffect|TRUE', #initial affect with random effects
  DRIFT = 'stateDependence',
  DIFFUSION = 'systemNoise')
```

Checking and understanding model specification can be much easier when looking at the full, expanded model equations in matrix form – we can generate a LaTeX formatted representation of the equations we just specified by running the **ctModelLatex()** function. To do this we need to install the **tinytex** package, if we have not already done so. Sometimes there are difficulties getting LaTeX compilation to work on various systems. For those who have difficulties, generally installing the R package **tinytex** via `install.packages('tinytex')`, then

running `tinytex::install_tinytex()`. In case of errors along the way, restarting R / Rstudio and clearing the workspace can help. For those that do not want the trouble of potential troubleshooting, the output can be found below.

Subject parameter distribution:	$\underbrace{[\text{initialAffect}_i]}_{\phi(i)} \sim \text{tform}\{\text{N}([\text{raw_initialAffect}], [\text{rawPCov_1_1}])\}$
Initial latent state:	$\underbrace{[\text{Affect}](t_0)}_{\eta(t_0)} \sim \text{N}\left(\underbrace{[\text{initialAffect}]}_{\text{TOMEANS}}, \underbrace{\text{UcorSDtoCov}\{[1e-06]\}}_{\text{Q}^*_{t_0}}\right)$
Deterministic change:	$\frac{d[\text{Affect}](t)}{d\eta(t)} = \left(\underbrace{[\text{stateDependence}]}_{\text{A}} \underbrace{[\text{Affect}](t)}_{\eta(t)} + \underbrace{[\text{B}]}_{\text{CINT}} \right) dt +$
Random change:	$\underbrace{\text{UcorSDtoChol}\{[\text{systemNoise}]\}}_{\text{G}} \frac{d[W_1](t)}{dW(t)}$
Observations:	$\underbrace{[\text{Affect}](t)}_{Y(t)} = \underbrace{[1]}_{\Lambda} \underbrace{[\text{Affect}](t)}_{\eta(t)} + \underbrace{[0]}_{\tau} +$ MANIFESTMEANS
Observation noise:	$\underbrace{[\text{residualSD}]}_{\Theta} \underbrace{[\epsilon_1](t)}_{\epsilon(t)}$ MANIFESTVAR
System noise distribution per time step:	$\Delta[W_{j \in [1,1]}](t-u) \sim \text{N}(0, t-u)$
	Observation noise distribution: $[\epsilon_{j \in [1,1]}](t) \sim \text{N}(0, 1)$

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.

We can now fit the model and ask for a summary of the parameter estimates.

```
ct_fit <- ctStanFit(datalong = data, ctstanmodel = ct_model) #fit the model to our data

summary(ct_fit, parmatrices= FALSE) # print summary of the fit, some output disabled
```

```

$residCovStd
  Affect
Affect  0.083

$resiCovStdNote
[1] "Standardised covariance of residuals"

$popsd
      mean     sd   2.5%   50% 97.5%
initialAffect 1.9348 0.3071 1.3896 1.9224 2.5855

$popmeans
      mean     sd   2.5%   50% 97.5%
initialAffect 5.4051 0.4224 4.5562 5.4170 6.1936
stateDependence -0.0950 0.0067 -0.1084 -0.0948 -0.0832
systemNoise     0.2007 0.0125 0.1776 0.2002 0.2261
residualSD      0.0535 0.0255 0.0193 0.0481 0.1190
B              0.9584 0.0543 0.8553 0.9593 1.0633

$popNote
[1] "covariance pars in sd / unconstrained cor form, see $parmatrices for cor/cov."

$loglik
[1] 32.14272

$npars
[1] 6

$aic
[1] -52.28544

$logposterior
[1] 32.14272

$parmatNote
[1] "For additional summary matrices, use argument: parmatrices = TRUE"

```

The summary output provides several key sections for interpreting model fit and parameters:

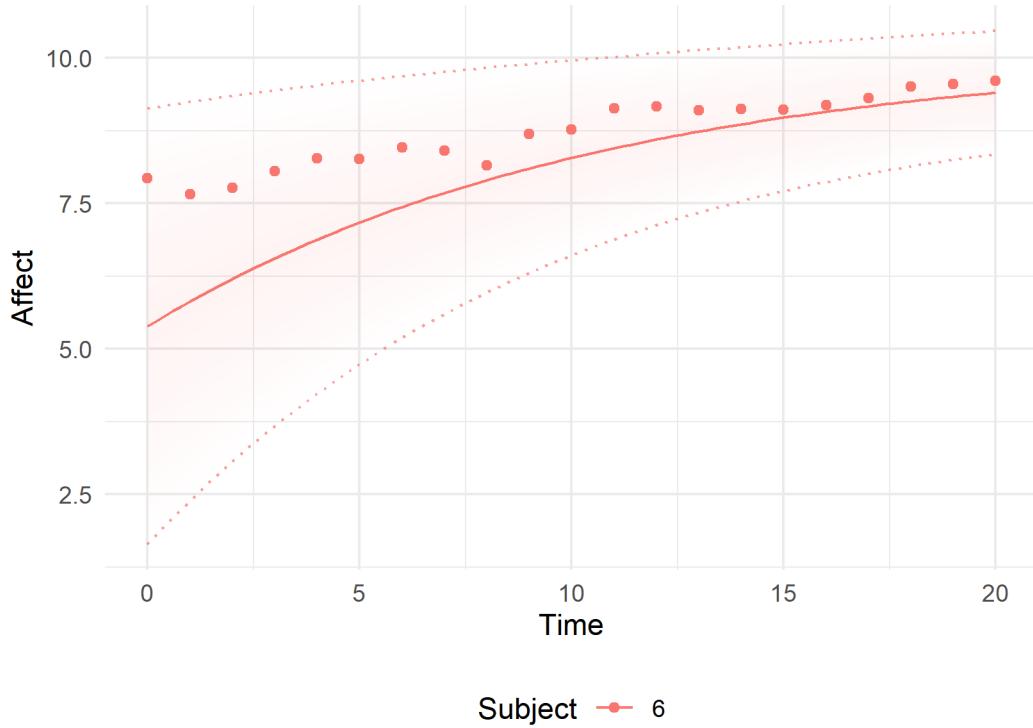
- **\$residCovStd**: Standardized residual covariance, indicating how well the model explains the data
- **\$popsd**: Population standard deviations for random effects in the model
- **\$popmeans**: Population-level parameter estimates for fixed effects model parameters
- **\$loglik**: Log-likelihood value for model comparison
- **\$aic**: Akaike Information Criterion for model selection

From our summary table, we can see that the estimated coefficients are not exactly the same as the true values, this is due to sample variation in the data. But the model captures the general trend of affect dynamics over time. The estimated state dependence coefficient is around -0.1, the continuous intercept (B) is approximately 1, and the system noise standard deviation is around 0.2. The random effects standard deviation for initial affect is approximately 2, and the residual standard deviation is around 0.05.

Visualizing Predictions

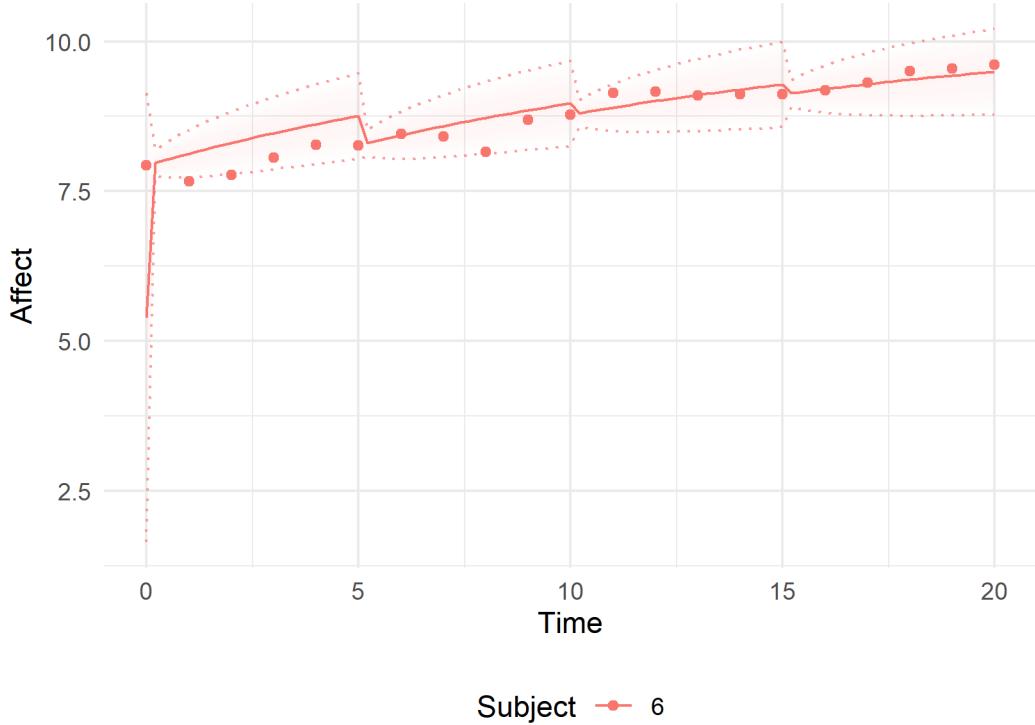
Using `ctsem` we can easily plot the model predictions and associated uncertainty. Our predictions can be conditional on all, none, or some of the individual subjects' observed data. First let's look at the predictions based solely on the parameter estimates, for an arbitrary subject, number 6 (multiple subjects can be specified in the `subjects` argument).

```
ctKalman(fit= ct_fit, plot = TRUE, subjects = 6, removeObs = TRUE)
```



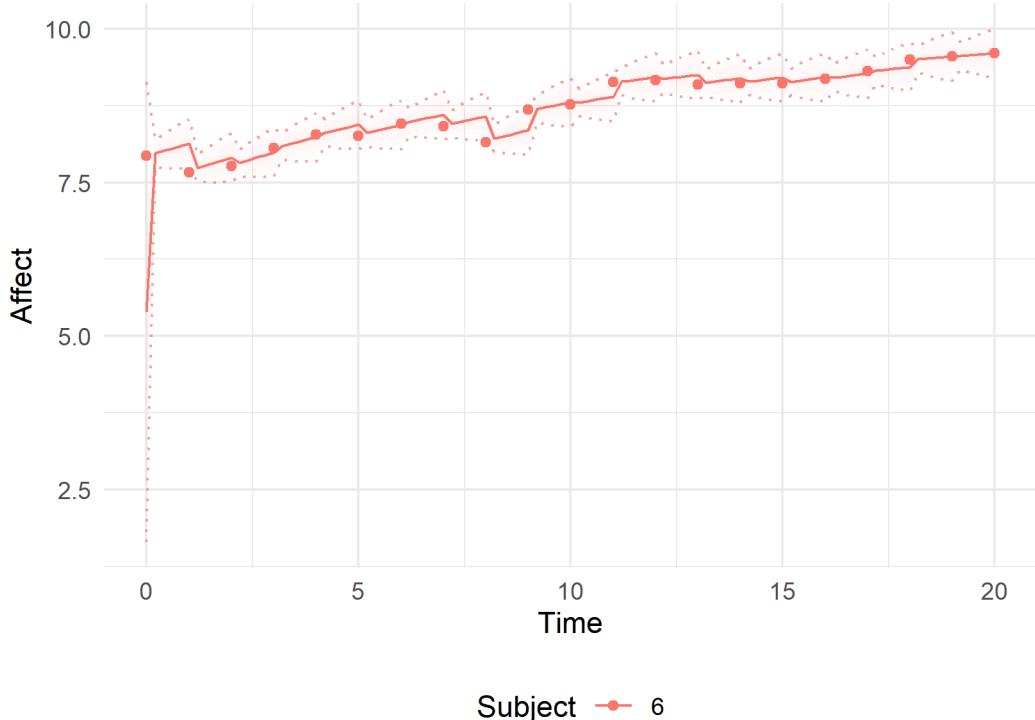
Just like in the case of our simple linear model, we see that the model does a poor job at predicting the observations (red dots) and there is a large degree of uncertainty (shaded red ribbon) around the model-implied trajectory (solid red line). Now let's allow the model to learn from every 5th observation to supplement the predictions it makes using the model estimates alone.

```
ctKalman(fit= ct_fit, plot = TRUE, subjects = 6, removeObs = 5)
```



We can see that the model already does a better job at predicting the observations, and the uncertainty in its predicted trajectory is reduced (though growing larger again towards the end of the time period before a new observation is fed into the model). Lastly, let us include all past and present observations (the default) and see how our model performs.

```
ctKalman(fit= ct_fit, plot = TRUE, subjects = 6, removeObs = FALSE)
```



Now the model does a fairly good job at predicting the observations, but there is still some uncertainty in its predictions.

Couple Dynamics with Self and Partner Effects (Continuous Time)

Until now, we have looked at Jill's affect in isolation. But she is going to couples therapy, which makes it very likely that how her partner, Bert, feels has a pronounced impact on how Jill feels. In the same way, Jill's affect is also likely to have a reciprocal impact on Bert's affect. This creates an interdependence between Jill and Bert's affect dynamics – a fundamental concept in dyadic data analysis where the behavior or state of one partner influences and is influenced by the other partner, creating a complex system of mutual dependencies. The statistical challenges of dyadic interdependence ([Kenny, Kashy, and Cook 2006](#)) are particularly relevant here as we transition from modelling individual dynamics to modelling coupled systems where cross-partner effects and shared environmental influences must be explicitly accounted for. We can model this interdependence in two ways:

1. We can introduce *a cross-effect state dependence term*, which reflects how Jill's affect at a given moment can have a direct effect on how Bert's affect changes at that moment. This is essentially the state dependence we modeled within a person, but now the state

(affect level) that drives the rate of change is also coming from another person. We can also introduce another cross-effect state-dependence term, capturing the influence of Bert's current state of affect on Jill's rate of change at that moment.

2. A *Common system noise term*, random life events that cause fluctuations in Jill's affect trajectory are also likely to contemporaneously impact Bert's affect (and vice versa). We can model factors that simultaneously shift our couple's rate of change, by allowing a random noise term for each partner, but allowing some common effect – that is, Jill and Bert's random affect fluctuations may be correlated.

The revised model which includes Jill and Bert's affect dynamics along with their interdependence can be written exactly as earlier (the dimensions of the model components will differ however), or as two equations, one for each partner:

$$\begin{aligned} d\eta_1 &= (A_1\eta_1(t) + A_{cross_{12}}\eta_2(t) + B_1)dt + G_{11}dW_1(t) + G_{12}dW_2(t) \\ d\eta_2 &= (A_2\eta_2(t) + A_{cross_{21}}\eta_1(t) + B_2)dt + G_{21}dW_1(t) + G_{22}dW_2(t) \end{aligned}$$

We have two equations, one for each partner's affect. The equation for $d\eta_1$ (rate of change of Jill's affect), implies that her rate of change depends on (1) her own current affect level ($A_1\eta_1(t)$ – the auto-effect), (2) her partner Bert's current affect level ($A_{cross_{12}}\eta_2(t)$ – the cross-effect from partner), (3) a constant input (B_1), and (4) random noise that may affect both partners ($G_{11}dW_1(t) + G_{12}dW_2(t)$). The equation for $d\eta_2$, rate of change in Bert's affect, implies that similarly, Bert's rate of change depends on his own affect ($A_2\eta_2(t)$), Jill's affect ($A_{cross_{21}}\eta_1(t)$), his constant (B_2), and random noise that may affect both partners ($G_{21}dW_1(t) + G_{22}dW_2(t)$). The cross-effects ($A_{cross_{12}}$ and $A_{cross_{21}}$) capture how partners influence each other's rate of change. If $A_{cross_{12}}$ is positive, when Bert's affect is high, Jill's affect is positively influenced – it either grows faster or declines slower.

Simulation

We will generate data for 20 couples. Each of the 40 people will have their own initial level of affect, but all dynamics and growth terms will be fixed across individuals for now.

Couple dynamics. To generate data for each couple, we can build on the previous model of affect that we used to represent the affect of one person. The first thing we need to add, is an equation to represent the affect process of the additional partner. Then we need to add our cross-effect and common-noise terms to the (differential) equations. In the code chunk below, we can see that we now have two equations to approximate two continuous affect processes, which are differentiated using `Affect1` (for partner 1) and `Affect2` (for partner 2). Apart from duplicating the equation for one person, each equation has *two* new elements to capture interdependency in affect dynamics. (1) The change in affect for each person depends on their partner's level of affect at the same time point. The coefficient of this cross-effect is denoted by

Across (e.g. `dAffect1 <- A*Affect1State + Across * Affect2State + B`). (2) We allow the random fluctuations in the couple's affect process to covary. We do this by adding an additional source of random noise `systemNoiseCrossState` to each person's model equation which is scaled by a common system noise coefficient (`Gcross`).

```
# Generate data for multiple subjects with individual differences
NSubjects <- 20
times <- seq(from=0, to=40, by=1) #generate sequence of time points when subjects are measured
Nobs <- length(times) #number of observations per subject
initialAffect1 <- rnorm(n = NSubjects, mean = 5, sd = 2)
initialAffect2 <- rnorm(n = NSubjects, mean = 5, sd = 2)
A <- -.2 #continuous time state dependence
Across <- .1 #cross-effect state dependence
B <- 1 #continuous intercept
G <- .2 #unique system noise coefficient
Gcross <- .1 #common system noise coefficient

#create empty data.frame to fill step by step
data <- data.frame(Subject= rep(NA,NSubjects*Nobs),
Time = rep(NA,NSubjects*Nobs),
Affect1 = rep(NA,NSubjects*Nobs),
Affect2 = rep(NA,NSubjects*Nobs)) #now with affect for two individuals

Nsteps <- 100 #number of steps in time to compute between each observation (increased precision)

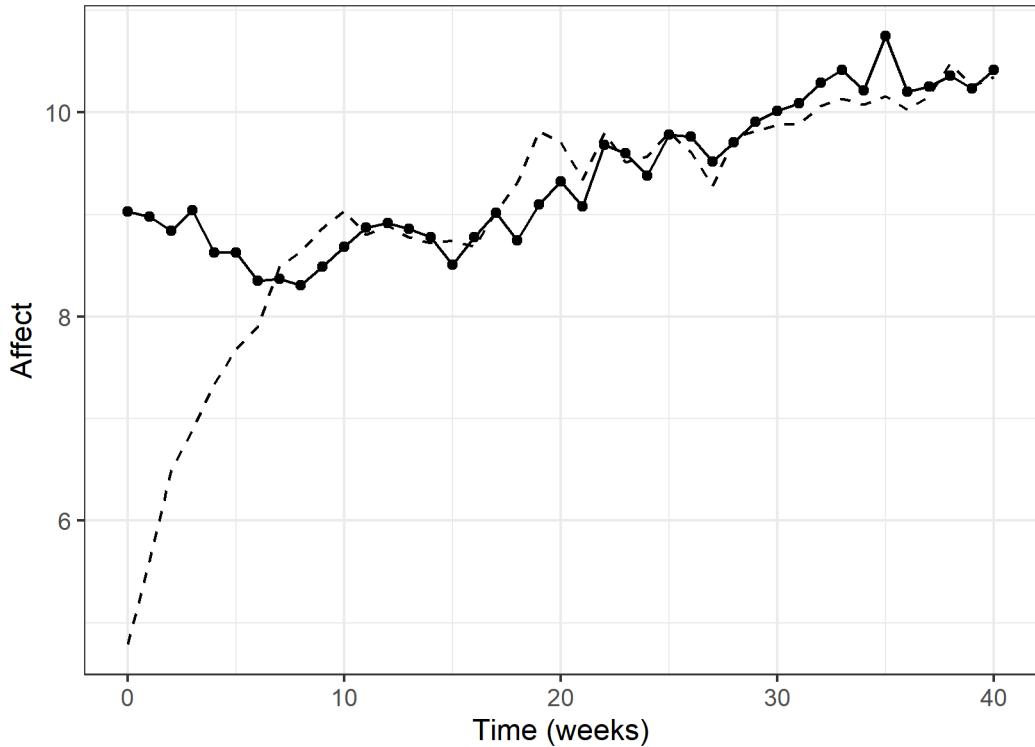
row <- 0 #initialize row counter, to track which row of the data.frame we are on
for(subi in 1:NSubjects){
  for(obsi in 1:Nobs){ #for each observation of a subject
    row <- row + 1
    if(obsi==1){
      Affect1State <- initialAffect1[subi] #if first time point, set to initial affect
      Affect2State <- initialAffect2[subi]
    }
    if(obsi>1){ #else compute new affect state by taking a sequence of small steps in time
      for(stepi in 1:Nsteps){ #take Nsteps in time between each observation
        # compute deterministic slopes of affect at earlier time point
        dAffect1 <- A*Affect1State + Across * Affect2State + B
        dAffect2 <- A*Affect2State + Across * Affect1State + B

        systemNoiseState1 <- rnorm(n=1, mean=0, sd=sqrt(1/Nsteps)) #unique noise for subj 1
        systemNoiseState2 <- rnorm(n=1, mean=0, sd=sqrt(1/Nsteps)) #unique noise for subj 2
        systemNoiseCrossState <- rnorm(n=1, mean=0, sd=sqrt(1/Nsteps)) #common noise

        Affect1State <- Affect1State + dAffect1 * 1/Nsteps + #update state using slope and time step
          G * systemNoiseState1 + Gcross * systemNoiseCrossState #and add unique and common noise
        Affect2State <- Affect2State + dAffect2 * 1/Nsteps + #update state using slope and time step
          G * systemNoiseState2 + Gcross * systemNoiseCrossState #and add unique and common noise
      }
    }
    data$Affect1[row] <- Affect1State #input affect data
    data$Affect2[row] <- Affect2State #input affect data
    data$Time[row] <- times[obsi] #input time data
    data$Subject[row] <- subi #input subject data
  }
}

data$Affect1 <- data$Affect1 + rnorm(n=nrow(data), mean = 0, sd = .05) #add measurement error
data$Affect2 <- data$Affect2 + rnorm(n=nrow(data), mean = 0, sd = .05) #add measurement error
```

```
plot_trajectory(data[data$Subject==1], y_var = "Affect1", y2_var = "Affect2")
```



Here we can see the growth process for one couple. Each partner's affect starts from a very different initial affect state, but quite quickly converges to a very similar trajectory. This can be partly attributed to the influence our couple has on each other's rate of growth over time. We can also see a similar pattern of fluctuations in their affect, around their overall level of growth. This is can be due to a set of common experiences the couple goes through, that impact their affect similarly. You can tweak the model parameters to see how each changes the generated trajectory.

Model Fitting

To understand the `ctsem` syntax for our multivariate model (one latent process and one observed variable for each partner) it helps to see the model equation in expandedmatrix form to more easily link it with the more compact syntax of `ctsem`. So our latent dynamic model for a couple:

$$d\eta_1 = (A_1 \eta_1(t) + A_{cross_{12}} \eta_2(t) + B_1)dt + G_{11}dW_1(t) + G_{12}dW_2(t)$$

$$d\eta_2 = (A_2\eta_2(t) + A_{cross_{21}}\eta_1(t) + B_2)dt + G_{21}dW_1(t) + G_{22}dW_2(t)$$

Can also be represented in expanded matrix form as:

$$\begin{pmatrix} d\eta_1 \\ d\eta_2 \end{pmatrix} = \left(\underbrace{\begin{pmatrix} A_1 & A_{cross_{12}} \\ A_{cross_{21}} & A_2 \end{pmatrix}}_{\text{Drift}} \underbrace{\begin{pmatrix} \eta_1 \\ \eta_2 \end{pmatrix}}_{\eta(t)} + \underbrace{\begin{pmatrix} B_1 \\ B_2 \end{pmatrix}}_{\text{CINT}} \right) dt + \underbrace{\begin{pmatrix} G_{11} & G_{12} \\ G_{21} & G_{22} \end{pmatrix}}_{\text{Diffusion}} \underbrace{d \begin{pmatrix} W_1 \\ W_2 \end{pmatrix}}_{dW(t)}(t)$$

This matrix equation packs both partners' equations into one larger matrix equation. On the left, the vector $(d\eta_1, d\eta_2)$ contains the rates of change for both partners. On the right, the drift matrix shows how each partner's current affect influences each partner's rate of change. The diagonal elements (A_1, A_2) are auto-effects (how each person's own affect influences their own change), while off-diagonal elements ($A_{cross_{12}}, A_{cross_{21}}$) are cross-effects (how one partner affects the other's change). The CINT (continuous intercept) vector contains the constants (B_1, B_2) that are steady inputs for each partner. The Diffusion matrix controls the noise structure – diagonal elements (G_{11}, G_{22}) are each partner's unique noise, while off-diagonal elements (G_{12}, G_{21}) capture shared noise between partners (representing e.g., common experiences the couple goes through, that impact their affect similarly).

Now we can connect the matrix form of our latent dynamic model to the `ctsem` syntax:

1. The `DRIFT` argument, specifies the auto- (A) and cross-effect (A_{cross}) state-dependence terms. We call them `auto` and `cross`, respectively.
2. The `CINT` argument specifies the continuous intercepts ($B1$ and $B2$). We call them continuous intercept `B1` and `B2`, and only estimate a fixed effect (one value for all subjects). To switch off random effects we need to add `FALSE` behind `B1||`, i.e. `B1|FALSE`.
3. The `DIFFUSION` argument specifies the covariance matrix of the system noise, and is specified in terms of standard deviations on the diagonal and ‘unconstrained’ correlation parameters on the lower off-diagonal. These unconstrained correlation parameters are unfortunately somewhat complex to interpret directly, but offer useful flexibility for things like individual differences in the pattern of correlations. The standard deviation parameters are `systemNoise1` and `systemNoise2` and the correlation parameter is called `systemNoiseCross`.
4. The initial values of our latent processes, are treated as random variables with their own mean and variance parameters: $\eta_{t0} = \mu_{0i} + \epsilon_{0i}$, where $\epsilon_{0i} \sim \mathcal{N}(0, \sigma^2)$. The `TOMEANS` argument, is used to specify the mean parameter for each partner's initial level of affect, which we call `initialAffect1` and `initialAffect2`. By adding `TRUE` behind `initialAffect||` we are allowing for random effects (between-subject variance) around the population estimate of the initial level of affect in our sample.

The latent process model is linked to our observations using a (linear) measurement model:

$$\begin{pmatrix} Affect_1 \\ Affect_2 \end{pmatrix}(t) = \underbrace{\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}}_{\Lambda} \underbrace{\begin{pmatrix} \eta_1 \\ \eta_2 \end{pmatrix}(t)}_{\eta(t)} + \underbrace{\begin{pmatrix} 0 \\ 0 \end{pmatrix}}_{\tau} + \underbrace{\begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{pmatrix}}_{\Theta} \underbrace{\begin{pmatrix} \epsilon_1 \\ \epsilon_2 \end{pmatrix}(t)}_{\epsilon(t)}$$

This equation links what we actually observe ($Affect_1(t)$, $Affect_2(t)$) to the underlying latent processes ($\eta_1(t)$, $\eta_2(t)$) that represent ‘true’ affect. Λ is a matrix of regression weights used to link the latent process to our observations, and is here set to the identity matrix (1s on diagonal, 0s off-diagonal), meaning each observed affect measure directly reflects its corresponding latent affect with a 1-to-1 relationship (no scaling or transformation). τ reflects a constant that can be used to shift the relationship between observations and a latent variable, but here is a zero vector, meaning there’s no systematic bias or offset in measurements. The Θ matrix contains measurement error standard deviations (σ_1 , σ_2) – these scale the disturbance terms ϵ to capture the idea that our observations are imperfect indicators of the true latent affect state, but that there is no correlation between the imperfections of each – the off-diagonal elements are set to 0.

1. The **LAMBA** option specifies the Λ matrix. Now it is a 2x2 matrix with the value 1 on the diagonal, meaning that the latent Affect (η) is directly and equally reflected in the observed Affect without any scaling.
2. The **MANIFESTMEANS** option specifies the measurement intercept (τ) which is simply a constant that is added to the measurement model to shift the relationship between observations and a latent variable. In this case it is set to zero, implying that there is no systematic offset in the measurement process.
3. The **MANIFESTVAR** option specifies the standard deviation (Θ) of the measurement error terms (ϵ), which we call **residualSD1** and **residualSD2**.

```
# Fit continuous time structural equation model
ct_model <- ctModel( #define the ctsem model
  manifestNames = c("Affect1", "Affect2"), #names of observed variables in dataset
  latentNames = c("Affect1", "Affect2"), #names of latent processes
  time = 'Time', #name of time column in dataset
  id = 'Subject', #name of subject column in dataset
  type='ct', #use continuous time / differential equation model (dt for discrete-time)
  MANIFESTVAR = c(
    'residualSD1', 0,
    0, 'residualSD2'), #sd of the residual / measurement error
  LAMBDA = diag(1,2), #relating latent process to observed variables
  MANIFESTMEANS=0, #no measurement intercept (1 observed variable relates directly to latent)
  CINT=c('B1||FALSE', 'B2||FALSE'), #continuous intercept with *no* random effects
  TOMEANS=c('initialAffect1||TRUE', 'initialAffect2||TRUE'), #initial affect with random effects
  DRIFT = c(
    'auto1', 'cross12', #auto effect for subj 1 and cross-effect from 2 to 1
    'cross21', 'auto2' ), #cross-effect from 1 to 2 and auto effect for subj 2
  DIFFUSION = c(
    'systemNoise1', 0, #system noise for subj 1, 0 in upper triangle (correlation only needs 1 par)
    'systemNoiseCross', 'systemNoise2')) #correlation in system noise, and sd for subj 2
ctModelLatex(ct_model) #generate LaTeX representation of the model
```

Below we generate the model equations, for a clearer view of the entire model structure.

Subject parameter distribution:

$$\underbrace{\begin{bmatrix} \text{initialAffect1}_i \\ \text{initialAffect2}_i \end{bmatrix}}_{\phi(i)} \sim \text{tform} \left\{ N \left(\begin{bmatrix} \text{raw_initialAffect1} \\ \text{raw_initialAffect2} \end{bmatrix}, \begin{bmatrix} \text{rawPCov_1..1} & \text{rawPCov_2..1} \\ \text{rawPCov_2..1} & \text{rawPCov_2..2} \end{bmatrix} \right) \right\}$$

Initial latent state:

$$\underbrace{\begin{bmatrix} \text{Affect1} \\ \text{Affect2} \end{bmatrix}}_{\eta(t_0)} (t_0) \sim N \left(\underbrace{\begin{bmatrix} \text{initialAffect1} \\ \text{initialAffect2} \end{bmatrix}}_{\text{TOMEANS}}, \underbrace{UcorSDtoCov \left\{ \begin{bmatrix} 1e-06 & 0 \\ 0 & 1e-06 \end{bmatrix} \right\}}_{\mathbf{Q}^*_{t_0} \text{TOVAR}} \right)$$

Deterministic change:

$$d\underbrace{\begin{bmatrix} \text{Affect1} \\ \text{Affect2} \end{bmatrix}}_{d\eta(t)} (t) = \underbrace{\begin{bmatrix} \text{auto1} & \text{cross12} \\ \text{cross21} & \text{auto2} \end{bmatrix}}_{\mathbf{A} \text{DRIFT}} \underbrace{\begin{bmatrix} \text{Affect1} \\ \text{Affect2} \end{bmatrix}}_{\eta(t)} (t) + \underbrace{\begin{bmatrix} \text{B1} \\ \text{B2} \end{bmatrix}}_{\mathbf{b} \text{CINT}} dt +$$

Random change:

$$\underbrace{UcorSDtoChol \left\{ \begin{bmatrix} \text{systemNoise1} & 0 \\ \text{systemNoiseCross} & \text{systemNoise2} \end{bmatrix} \right\}}_{\mathbf{G} \text{DIFFUSION}} d\underbrace{\begin{bmatrix} W_1 \\ W_2 \end{bmatrix}}_{d\mathbf{W}(t)} (t)$$

Observations:

$$\underbrace{\begin{bmatrix} \text{Affect1} \\ \text{Affect2} \end{bmatrix}}_{\mathbf{Y}(t)} (t) = \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}}_{\Lambda \text{LAMBDA}} \underbrace{\begin{bmatrix} \text{Affect1} \\ \text{Affect2} \end{bmatrix}}_{\eta(t)} (t) + \underbrace{\begin{bmatrix} 0 \\ 0 \end{bmatrix}}_{\boldsymbol{\tau} \text{MANIFESTMEANS}} +$$

Observation noise:

$$\underbrace{\begin{bmatrix} \text{residualSD1} & 0 \\ 0 & \text{residualSD2} \end{bmatrix}}_{\Theta \text{MANIFESTVAR}} \underbrace{\begin{bmatrix} \epsilon_1 \\ \epsilon_2 \end{bmatrix}}_{\epsilon(t)} (t)$$

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

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

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.

Fit and Summarize ctsem Model

Then we fit our model and could, as earlier, use **summary** to view estimated parameters. However, since it is difficult to get a feeling for the nonlinear model-implied trajectory of a system

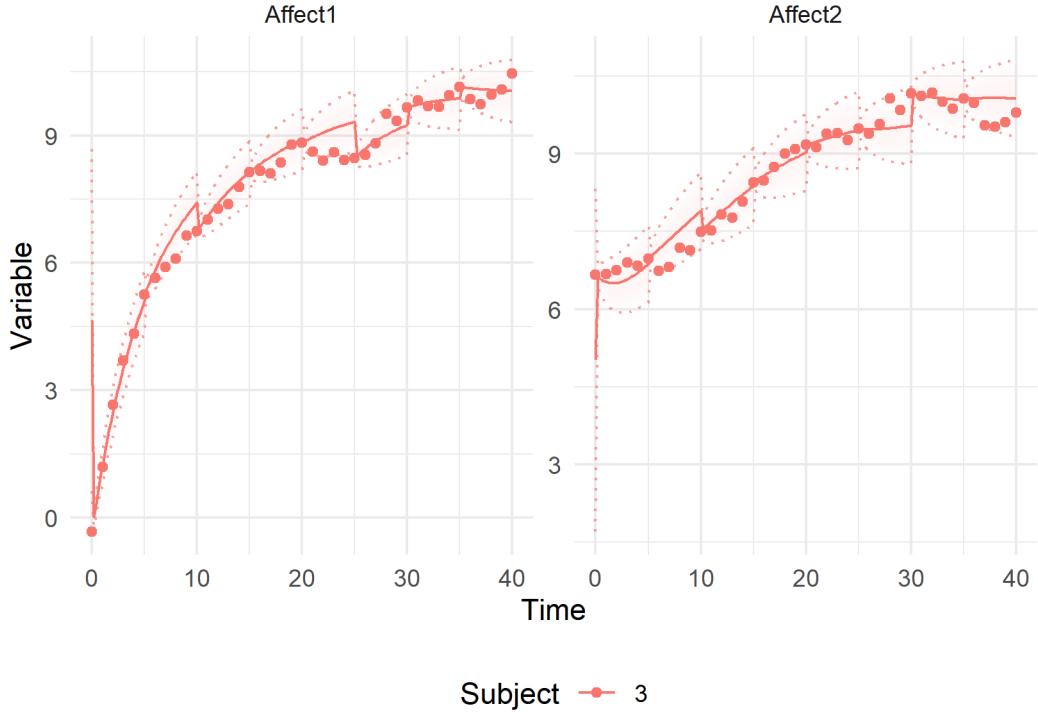
composed of multiple interdependent process through numerical values, we will focus on a visual representation of our system instead for the time being.

```
ct_fit <- ctStanFit(datalong = data, ctstanmodel = ct_model) #fit the model to our data
```

Visualize Predictions

Just like before, we can look at how well the model predicts our observations based on the estimated model parameters plus every 5th data point for our 3rd dyad (subject 3). When we visualize predictions from our model, now including multivariate dynamics, state dependence, and random fluctuations, we see a very different picture to the earlier linear models. Our model is able to capture a reality where there is a complex interplay between the nonlinear affect dynamics of two individuals. The uncertainty in the predictions reflects two sources: The first, is that there are inherently unpredictable fluctuations in affect due to factors we have not observed or modeled. So our model is inevitably imperfect. The second, is that each measurement of affect is likely an imperfect indicator of the underlying affect state, so we can't be sure of the true state of the system even at the moment where we measure it (via e.g., a survey on a smartphone). We could visualise the latent process predictions and uncertainty by changing the `kalmanvec` argument to '`etaprior`' instead of '`yprior`' (which shows predictions for observations at each moment).

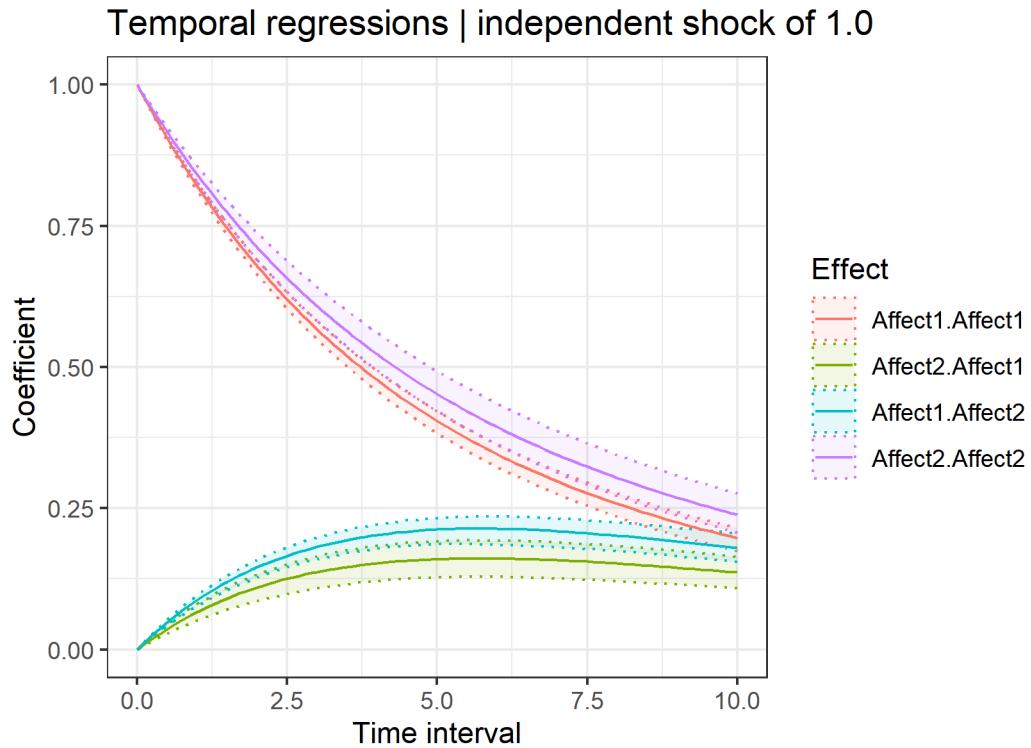
```
ctKalman(fit = ct_fit, plot=TRUE, subjects = 3, kalmanvec=c('y','yprior'), removeObs = 5)
```



Visualize Dynamics – Independent Shocks

We can also use visualizations to build intuition about the model-implied dynamics (i.e. auto-effect and cross-effect state dependencies). To do this, we can create a plot that tracks the results of an imaginary experiment. In this experiment, we take both Jill and Bert, and place them in a completely controlled setting, where no external factors can impact their affect. That is, their affect remains at baseline unless we do something about it. Second, we intervene on Jill's affect to increase it by 1 at time 0. Third, we plot how Jill's and Bert's affect would be expected to change over 10 weeks following the shock to Jill, as a function of their individual auto-effects and the cross-effects they have on each other. For example, if we increase Jill's affect by 1 at time 0, her affect is expected to still be approximately 0.40 above baseline after 5 weeks (red solid line, `Affect1.Affect1` in the plot at time interval = 5) because of her self-dependency and the bidirectional cross-effects between her and Bert. Instead, Bert's affect doesn't jump initially, but rises more gradually in response to the higher affect of Jill – it is expected to be 0.20 by week 5 (green solid line, `Affect2.Affect1` in the plot at time interval = 5). We can also visualize what happens if we increase Bert's affect by 1 *instead of Jill's* by viewing the blue and purple solid lines. This plot is known as an impulse response function plot ([Lütkepohl 2005](#)).

```
ctStanDiscretePars(ct_fit,plot=T) #plot dynamics
```



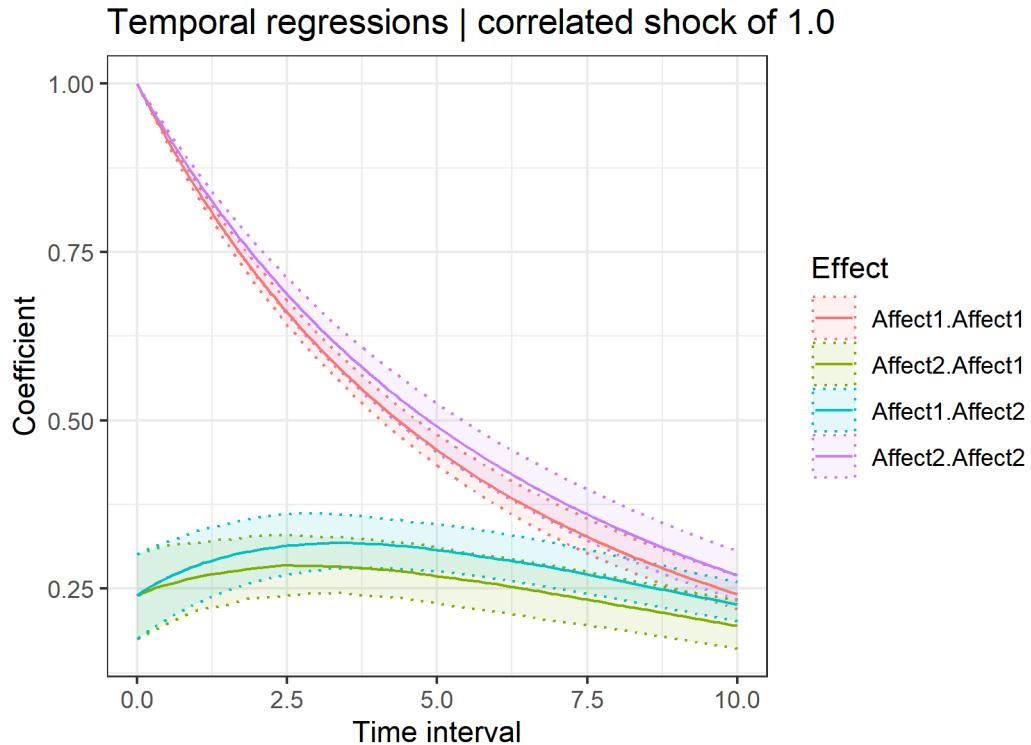
Visualize Dynamics – Correlated Shocks

Our hypothetical experiment with independent shocks, may have low generalizability to real world dynamic systems. This is because often a shock that happens to one partner also affects the other partner, and even though we may model this correlation, we have only been able to estimate model parameters in this correlated context – the impulse response plot (and direct interpretation of state dependence coefficients) may not generalise well to the hypothetical intervention scenario depicted above (see [Driver 2023](#), for more on the combined interpretation of system noise and dynamics).

One way to consider this issue is to create a plot where we consider the estimated correlation in the system noise, which tells us to what extent Jill and Bert's fluctuations are related, and incorporate this into the initial impulse – depicting essentially a more ‘typical’ impulse for the estimated system. This essentially means that when Jill experiences a random fluctuation in affect, Bert is also likely to experience a fluctuation, depending on the estimated correlation. Instead of plotting an impulse response function based solely on an independent 1-unit shock, we modify the impulse response function to reflect that a shock to one partner comes with a

correlated fluctuation in the other partner's affect. For example, if the system noise correlation is estimated at 0.25, then when Jill's affect increases by 1 unit, Bert's affect is expected to change by approximately 0.25 units as a result of that shared noise component. With `ctsem` we can simulate and plot the impulse response function, such that when one partner is shocked by 1 unit the response in the other partner is scaled by the system noise correlation. This gives a more realistic picture of the dynamic interplay between the partners.

```
ctStanDiscretePars(ct_fit,plot=T, observational=TRUE) #plot dynamics
```



From this plot we see that when Jill's affect increases by 1 at T0, Bert's affect is expected to contemporaneously fluctuate in the same direction as well (positive system noise correlation). The positive cross-effect coefficients also show that there is a positive bidirectional relationship between the two partners. Such that when one partner's affect increases we can expect that to also cause a subsequent increase in the other partner's affect. Here, we can be sure that this reflects a causal effect because we generated the data with such an effect. But in a real-world scenario, we would need to consider the possibility of confounding variables that could explain this estimated relationship. The system noise correlation can account for confounders that change very quickly (compared to the speed of change of affect), but is unlikely to sufficiently account for confounders that change at a similar speed to affect. For instance, a sudden loud noise that startles both partners will lead to a very brief fluctuation in affect for both

partners, which could be reasonably reflected in the system noise. However, a prolonged period of gloomy weather might cause a gradual systematic shift in the affect of both partners that can be misinterpreted as evidence that one partner's affect directly impacts the other's affect. Such slowly changing confounders are not accounted for in this model at present, but allowing for stable differences (in e.g., the continuous intercept) is one way that very slow / stable confounders can be accounted for – more on this later.

Individual Differences in System Dynamics

In the above models, we have generally assumed that all individuals have the same system dynamics. In reality, people may have highly heterogeneous system dynamics. This is because there are likely to be an immense number of factors that contribute to what we might consider as the individual's dynamic system. Factors that change slowly or not at all, with respect to our observed time window, can reasonably be treated as stable individual differences. Such individual differences may exist in the magnitude of random fluctuations, with some people having more stable affect systems while others have a more volatile affect. Alternatively, in the case of couples, some dyads may have more interdependent affect dynamics with stronger coupling, while other dyads are more independent.

Couple Dynamics with Individual Differences and Time-Independent Moderation (Continuous Time)

For our case, we are going to consider two sources of individual differences. First, we are going to assume that individuals have differences in their long-term, post therapy baseline affect. This will be done by allowing individual differences in the continuous intercept. Allowing for this makes our model a continuous-time analogue to the popular random intercept cross-lagged-panel model ([Hamaker, Kuiper, and Grasman 2015](#)). Second, we will assume that couples who have been together longer have a stronger influence on each others' affect. This will be incorporated by allowing for individual differences in the strength of our cross-effect state-dependence terms, and by allowing the size of each couple's cross-effect to depend on the average time they spend together.

Simulation

The code chunk to run this simulation, builds on the code we used earlier. To generate a continuous intercept for each person we sample from a normal distribution, `B1 <- rnorm(n = NSubjects, mean = 2, sd = .3)`. We assume that people with a higher affect are more likely to get in a relationship, thus we add a common value sampled from a normal distribution to the value for each person in a dyad. That is, `Bcommon <- rnorm(n = NSubjects, mean`

$= 1$, $sd = .2$) is added to the sampled values of B1 and B2 ($B1 \leftarrow Bcommon + rnorm(n = NSubjects, mean = 2, sd = .3)$).

To add individual differences in the continuous intercept parameter, we index the parameter to denote that it is specific to the denoted subject. So B1 becomes B1[subi] where subi refers to the subject identifier.

A similar process is used to generate data for our heterogeneous cross effects. We sample a cross-effect for each couple from a normal distribution and then add the time the couple has spent together, scaled by a coefficient that determines the degree of influence time spent together has on each couple's cross effects $Across \leftarrow rnorm(n = NSubjects, mean = .1, sd = .05) + .05 * TimeTogetherZ$. The cross-effect parameter is also indexed in our data generating loop to generate a subject-specific parameter, e.g. $Across[subi] * Affect2State$.

```
# Generate data for multiple subjects with individual differences
NSubjects <- 20
times <- seq(from=0, to=40, by=1) #generate sequence of time points when subjects are measured
Nobs <- length(times) #number of observations per subject
initialAffect1 <- rnorm(n = NSubjects, mean = 5, sd = 2)
initialAffect2 <- rnorm(n = NSubjects, mean = 5, sd = 2)
TimeTogether <- runif(n=NSubjects, min = 0, max = 20) #time together in years
TimeTogetherZ <- scale(TimeTogether) #standardise time together
A <- -.4 #continuous time state dependence
Across <- rnorm(n = NSubjects, mean = .1, sd = .05) + .05 * TimeTogetherZ #cross-effect state dependence

# generate continuous intercepts for each individual,
# assuming high affect individuals may partner with high affect individuals
Bcommon <- rnorm(n = NSubjects, mean = 1, sd = .2) #common continuous intercept variance
B1 <- Bcommon + rnorm(n = NSubjects, mean = 2, sd = .3) #unique continuous intercept for subj 1
B2 <- Bcommon + rnorm(n = NSubjects, mean = 2, sd = .3) #continuous intercept for subj 2
cor(B1,B2) #check if people with higher affect tend to couple
```

[1] 0.2729391

```
G <- .4 #unique system noise coefficient
Gcross <- .1 #common system noise coefficient

#create empty data.frame to fill step by step
data <- data.frame(Subject= rep(NA,NSubjects*Nobs),
                    Time = rep(NA,NSubjects*Nobs),
                    Affect1 = rep(NA,NSubjects*Nobs),
                    Affect2 = rep(NA,NSubjects*Nobs)) #now with affect for two individuals

Nsteps <- 100 #number of steps in time to compute between each observation (increased precision)

row <- 0 #initialize row counter, to track which row of the data.frame we are on
for(subi in 1:NSubjects){
  for(obsi in 1:Nobs){ #for each observation of a subject
    row <- row + 1
    if(obsi==1){
      Affect1State <- initialAffect1[subi] #if first time point, set to initial affect
      Affect2State <- initialAffect2[subi]
    }
    if(obsi>1){ #else compute new affect state by taking a sequence of small steps in time
      for(stepi in 1:Nsteps){ #take Nsteps in time between each observation
```

```

#compute deterministic slope of affect at earlier time point
dAffect1 <- A*Affect1State + Across[subi] * Affect2State + B1[subi]
dAffect2 <- A*Affect2State + Across[subi] * Affect1State + B2[subi]

systemNoiseState1 <- rnorm(n=1, mean=0, sd=sqrt(1/Nsteps)) #unique noise for subj 1
systemNoiseState2 <- rnorm(n=1, mean=0, sd=sqrt(1/Nsteps)) #unique noise for subj 2
systemNoiseCrossState <- rnorm(n=1, mean=0, sd=sqrt(1/Nsteps)) #common noise for both

#update states using slope and time step, and add unique and common system noise
Affect1State <- Affect1State + dAffect1 * 1/Nsteps +
  G * systemNoiseState1 + Gcross * systemNoiseCrossState

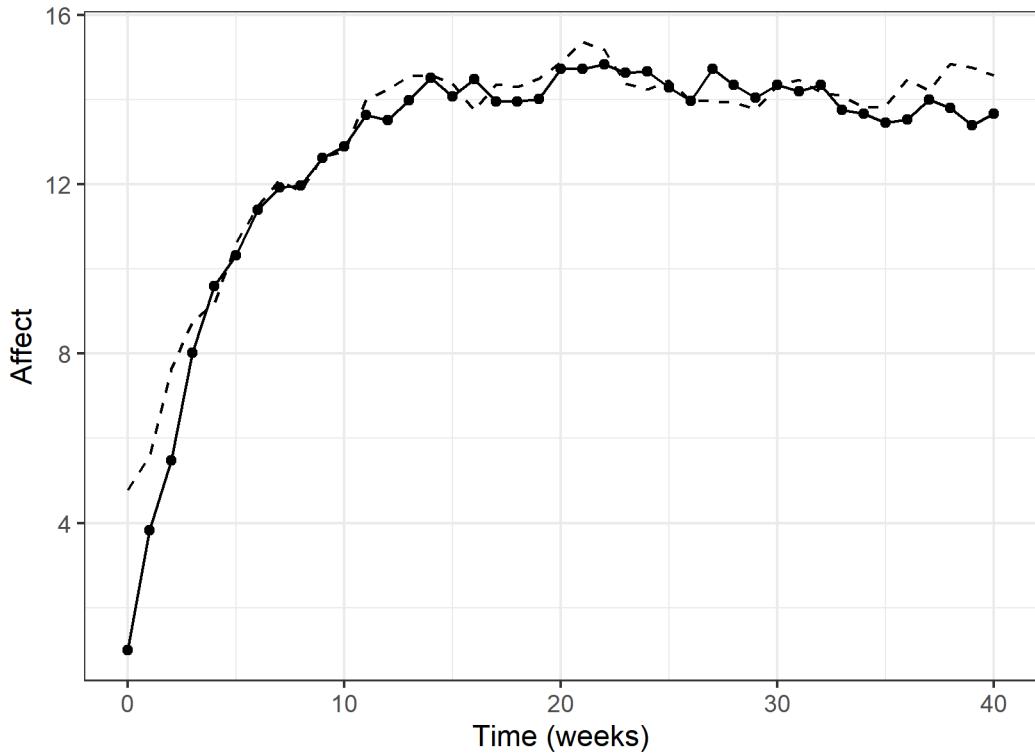
Affect2State <- Affect2State + dAffect2 * 1/Nsteps +
  G * systemNoiseState2 + Gcross * systemNoiseCrossState
}

data$Affect1[row] <- Affect1State #input affect data
data$Affect2[row] <- Affect2State #input affect data
data$Time[row] <- times[obsi] #input time data
data$Subject[row] <- subi #input subject data
data$TimeTogetherZ[row] <- TimeTogetherZ[subi] #input time together data
}
}

data$Affect1 <- data$Affect1 + rnorm(n=nrow(data), mean = 0, sd = .05) #add measurement error
data$Affect2 <- data$Affect2 + rnorm(n=nrow(data), mean = 0, sd = .05) #add measurement error

plot_trajectory(data[data$Subject==1], y_var = "Affect1", y2_var = "Affect2")

```



Model Fitting

These individual differences can be accommodated in `ctsem` using both ‘fixed’ and ‘random’ effect approaches. In the fixed effect approach, we rely on observed covariates (time-independent predictors in `ctsem`) to moderate the system parameters and explain the individual differences. In `ctsem` these are at present largely restricted to linear effects. For alternative forms, the usual approach of including higher order terms (e.g., age and age squared) is possible. Creative use of `ctsem`’s modelling flexibility could also be used to estimate other nonlinear effects.

In the random effect approach, we assume that the individual differences are at least partly due to unobserved factors, and estimate the variance and correlations in these factors, relying purely on the observed data (without covariates) to estimate where each individual’s parameter(s) sit with respect to their distribution of subject-specific deviations from the average population estimate.

By combining fixed and random effects, we try to improve the estimated model for each individual by allowing individuals to have unique system dynamics, but not ‘completely unique’ – we still rely to some extent on how other individuals behave (for more on continuous-time hierarchical modelling see [Driver and Voelkle 2018a](#)). In `ctsem` we can either decide to automatically specify random effects (preferable in most cases), or manually add random effects via the specification of additional latent processes (for details on the manual method see the [Driver and Tomasik 2023](#)). For our example, we will leave the specification of the random effects to the `ctsem` back-end, which estimates all random-effects as correlated.

So, let’s go over the new syntax in our `ctsem`. In terms of data that we fit our model on, we add the time couples spent together (`TimeTogetherZ`) as a stable source of individual differences (time-independent predictor). This is done using the argument `TIPredNames = c('TimeTogetherZ')`. By default when covariates are included in `ctsem` they moderate all parameters of the system. Thus, we turn the default moderation off using `tipredDefault = FALSE`. Now if we want our time-independent predictor to moderate a specific parameter, we need to explicitly specify the moderation. The moderation of a cross-effect (e.g. `cross21`) by `TimeTogetherZ` can be specified within the DRIFT matrix by inserting the name of the predictor to the fifth slot (slots are denoted by the pipe | operator, where each | defines 1 slot). Thus, we write `cross21 || TRUE || TimeTogetherZ`. This would estimate the `cross21` parameter, allow it to vary as a linear function of `TimeTogetherZ`, and allow for random effects (done by adding `TRUE` to the third slot, `cross21 || TRUE`) to account for any individual differences that could not be explained by `TimeTogetherZ`. It is important to note that moderators in `ctsem` can dramatically influence the interpretation of parameters and lead to confusion if care is not taken – usually centering, and possibly scaling, the moderator(s) is a good idea. Since we also want random effects for the continuous intercept of each partner we specify these using `CINT=c('B1 || TRUE', 'B2 || TRUE')`. All the other arguments are explained in prior sections of this tutorial, or can be found in the `ctsem` manual (<https://cran.r-project.org/package=ctsem/vignettes/hierarchicalmanual.pdf>).

```

# Fit continuous time structural equation model
ct_model <- ctModel( #define the ctsem model
  tipredDefault = FALSE, #moderation disabled unless explicitly specified
  TIpredNames = c('TimeTogetherZ'), #names of time independent predictors in dataset
  manifestNames = c("Affect1",'Affect2'), #names of observed variables in dataset
  latentNames = c("Affect1",'Affect2'), #names of latent processes
  time = 'Time', #name of time column in dataset
  id = 'Subject', #name of subject column in dataset
  type='ct', #use continuous time / differential equation model (dt for discrete-time)
  MANIFESTVAR = c(
    'residualSD1',0,
    0, 'residualSD2'), #sd of the residual / measurement error
  LAMBDA = diag(1,2), #relating latent process to observed variables
  MANIFESTMEANS=0, #no measurement intercept (1 observed variable relates directly to latent)
  CINT=c('B1||TRUE','B2||TRUE'), #continuous intercept with random effects
  TOMEANS=c('initialAffect1||TRUE','initialAffect2||TRUE'), #initial affect with random effects
  DRIFT = c(
    'auto1', 'cross21||TimeTogetherZ', #auto effect for subj 1 and cross-effect from 2 to 1
    'cross21||TimeTogetherZ','auto2' ), #cross-effect from 1 to 2 and auto effect for subj 2
  DIFFUSION = c(
    'systemNoise1', 0, #system noise sd for subj 1, 0 in upper triangle (1 par for correlation)
    'systemNoiseCross', 'systemNoise2')) #correlation in system noise, and sd for subj 2 noise

```

```
ct_fit <- ctStanFit(datalong = data, ctstanmodel = ct_model) #fit the model to our data
```

```
s = summary(ct_fit) #store summary of the fit
```

Now if we look at the `$tipreds` (time-independent predictors) section of the summary, we have an estimated value for the effect of time together on the cross-effect state dependence. We can see that the cross-effect state dependence increases by approximately .05 for each increase of 1 in standardized time together. Thus, partner's who spent more time together have more tightly coupled affect.

```
print(s$tipreds)
```

	mean	sd	2.5%	50%	97.5%	z
tip_TimeTogetherZ_cross21	0.0497	0.0105	0.0298	0.0496	0.0706	4.7163

The `$rawpopcorr` section of the summary shows the estimated correlations between the random effects of the model – the initial states and the continuous intercepts. We can see that the correlation between the random effects for the continuous intercepts (the `B2__B1` parameter) is approximately the true correlation we used when generating the data.

```
print(round(s$rawpopcorr,2))
```

	mean	sd	2.5%	50%	97.5%	z
initialAffect2__initialAffect1	0.16	0.20	-0.23	0.16	0.52	0.78
cross21__initialAffect1	-0.42	0.36	-0.87	-0.52	0.53	-1.16
B1__initialAffect1	-0.19	0.25	-0.65	-0.19	0.32	-0.74
B2__initialAffect1	0.18	0.27	-0.39	0.17	0.69	0.64

cross21__initialAffect2	-0.14 0.31 -0.63 -0.16 0.49 -0.45
B1__initialAffect2	-0.25 0.24 -0.68 -0.27 0.29 -1.04
B2__initialAffect2	-0.25 0.27 -0.68 -0.27 0.32 -0.94
B1__cross21	0.03 0.35 -0.63 0.04 0.63 0.10
B2__cross21	0.33 0.42 -0.66 0.45 0.88 0.80
B2__B1	0.19 0.28 -0.36 0.20 0.67 0.68

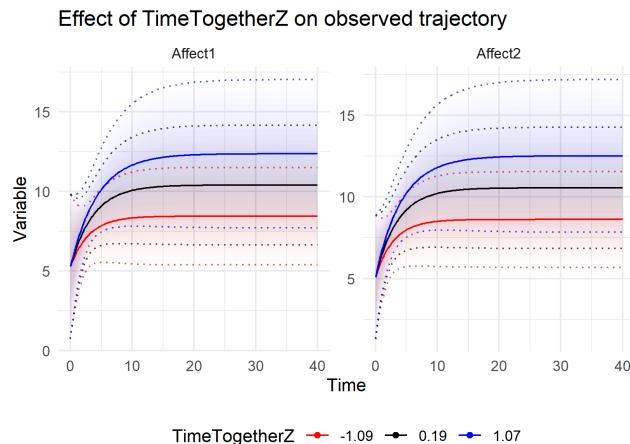
Visualize Individual Differences

We can plot the trajectory of affect implied by the estimated model parameters, conditional on values of -1, 0, and 1 on any time-independent predictors included in the model.

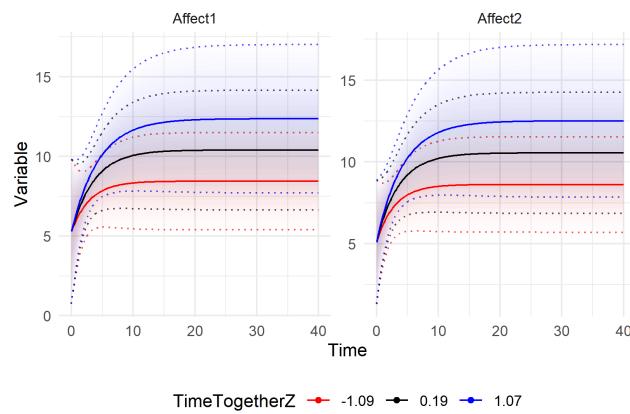
The first set of plots (**Effect of TimeTogetherZ on observed trajectory**), show the observed trajectory of partner 1 (Affect1) and partner 2 (Affect2) based on the three cutoffs of **TimeSpentTogetherZ** (-1.28, 0.01, 1.06). We can see that couples that spent more time together have a faster rate of growth in affect and also plateau at a higher affect value. The second set of plots (**Effect of TimeTogetherZ on latent trajectory**), shows the same information but for the latent trajectories of affect.

The third set of plots, (**Temporal regressions | independent shock of 1.0**) show an impulse response function for an independent 1-unit shock, but this time separated by our cutoffs of time spent together. Here we can visualize the fact that when a person experiences a shock to their affect (increase of 1) their partner is expected to change more over time if they spend more time together, on average. The fourth set of plots (**Temporal regressions | correlated shock of 1.0**), shows the impulse response function but this time taking into consideration the estimated correlated system noise. This function returns a list of **ggplot2** objects, which in some circumstances needs to be explicitly **print()**ed to show the plots.

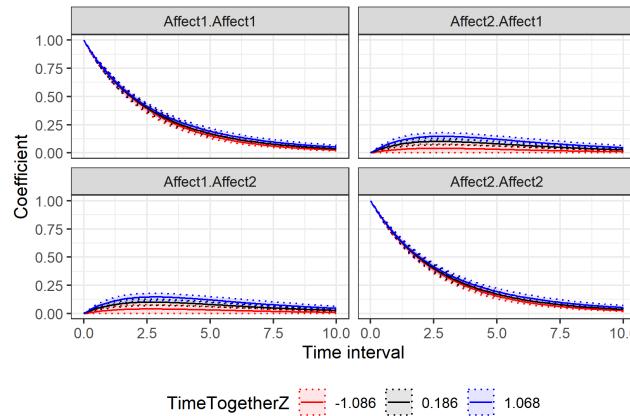
```
tipredplots <- ctPredictTIP(ct_fit, plot=T, tipreds = c('TimeTogetherZ'))
print(tipredplots)
```



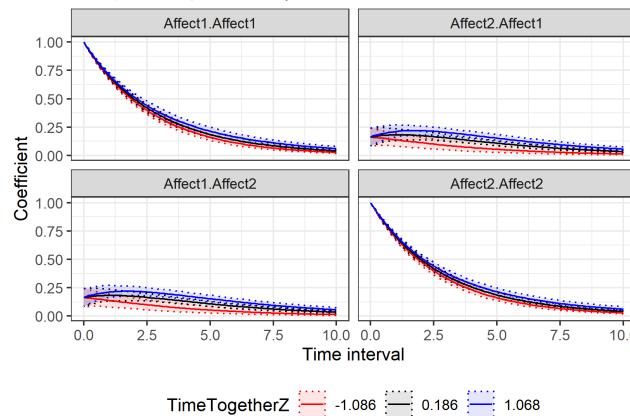
Effect of TimeTogetherZ on latent trajectory



Temporal regressions | independent shock of 1.0



Temporal regressions | correlated shock of 1.0



Couple Dynamics with Individual Differences and Time-Dependent Moderation (Continuous Time)

We may feel uncomfortable relying on people's retrospective estimate of how much time they spend together to moderate their degree of interdependence. However, we also do not want to assume that couples continuously affect each other even when they are not in each other's presence. For instance, according to our model, if Jill gets fired, her change in affect would immediately influence Bert's affect, even if Bert is completely unaware of this event. If we have information about Jill and Bert's proximity, we can use this information to tell our model to only allow Jill's affect to impact Bert's rate of change when the couple is together. This can be done by moderating our couple's cross-effect parameter by a time-dependent moderator of presence.

Simulation

To generate data for a model with a time-dependent moderator of cross-effect state dependence, we first need to generate a time-dependent moderator that indicates when a couple is communicating (0 no communication, 1 = communication). To do this, we will sample a value of 0 or 1 from a binomial distribution, where the probability of communication at each observation is given by the time couples spend together `Comm <- rbinom(n = NSubjects, size = 1, prob = TimeTogether / 20)`. Such that couples that spend more time together will have more observations when they are communicating. To sample a value for each observation `Nobs` per subject `Nsubjects`, we use the `lapply` function which is essentially a loop over each `TimeTogetherValue` (`t`), `Comm <- lapply(TimeTogether, function(t) rbinom(n = Nobs, size = 1, prob = t / 20))`. Then the communication value `comm_val` we use at each time step is simply the the observed `Comm` value during the current observation, `comm_val <- Comm[[subi]][obsi]`.

Second, we add the time-dependent moderator in our model of affect dynamics. Which makes the rate of change for partner 1 `dAffect1 <- A*Affect1State + Across[subi] * Affect2State * comm_val + B1[subi]`, where `Across` is multiplied by the time-dependent moderator `comm_val` at each time step. This makes it so there is no cross-effect when the couple is not communicating (`comm_val = 0`).

```
# Generate data for multiple subjects with individual differences
NSubjects <- 20
times <- seq(from=0, to=40, by=1) #generate sequence of time points when subjects are measured
Nobs <- length(times) #number of observations per subject
initialAffect1 <- rnorm(n = NSubjects, mean = 5, sd = 2)
initialAffect2 <- rnorm(n = NSubjects, mean = 5, sd = 2)
TimeTogether <- runif(n=NSubjects, min = 0, max = 20) #time together in years
TimeTogetherZ <- scale(TimeTogether) #standardise time together
A <- -.4 #continuous time state dependence
Across <- rnorm(n= NSubjects, mean = .1, sd=.05) #cross-effect state dependence
```

```

# Sample from a binomial distribution with prob based on time spent together
Comm <- lapply(TimeTogether, function(t) rbinom(n = Nobs, size = 1, prob = t / 20))

# Check the relationship: compute the mean interaction for each subject and correlate with TimeTogetherZ
Comm_means <- sapply(Comm, mean)
print(cor(Comm_means, TimeTogetherZ))

[,1]
[1,] 0.9755683

# generate continuous intercepts for each individual,
# assuming high affect individuals may partner with high affect individuals
Bcommon <- rnorm(n = NSubjects, mean = 1, sd = .2) #common continuous intercept variance
B1 <- Bcommon + rnorm(n = NSubjects, mean = 2, sd = .3) #unique continuous intercept for subj 1
B2 <- Bcommon + rnorm(n = NSubjects, mean = 2, sd = .3) #continuous intercept for subj 2
cor(B1,B2) #check if people with higher affect tend to couple

[1] 0.4976166

G <- .4 #unique system noise coefficient
Gcross <- .1 #common system noise coefficient

#create empty data.frame to fill step by step
data <- data.frame(Subject= rep(NA,NSubjects*Nobs),
Time = rep(NA,NSubjects*Nobs),
Affect1 = rep(NA,NSubjects*Nobs),
Affect2 = rep(NA,NSubjects*Nobs)) #now with affect for two individuals

Nsteps <- 100 #number of steps in time to compute between each observation (increased precision)

row <- 0 #initialize row counter, to track which row of the data.frame we are on
for(subi in 1:NSubjects){
  for(obsi in 1:Nobs){ #for each observation of a subject
    row <- row + 1
    comm_val <- Comm[[subi]][obsi]
    if(obsi==1){
      Affect1State <- initialAffect1[subi] #if first time point, set to initial affect
      Affect2State <- initialAffect2[subi]
    }
    if(obsi>1){ #else compute new affect state by taking a sequence of small steps in time
      for(stepi in 1:Nsteps){ #take Nsteps in time between each observation

        #compute deterministic slope of affect at earlier time point
        dAffect1 <- A*Affect1State + Across[subi] * Affect2State * comm_val + B1[subi]
        dAffect2 <- A*Affect2State + Across[subi] * Affect1State * comm_val + B2[subi]

        systemNoiseState1 <- rnorm(n=1, mean=0, sd=sqrt(1/Nsteps)) #unique noise for subj 1
        systemNoiseState2 <- rnorm(n=1, mean=0, sd=sqrt(1/Nsteps)) #unique noise for subj 2
        systemNoiseCrossState <- rnorm(n=1, mean=0, sd=sqrt(1/Nsteps)) #common noise for both individuals

        #update states using slope and time step, and add unique and common system noise
        Affect1State <- Affect1State + dAffect1 * 1/Nsteps +
          G * systemNoiseState1 + Gcross * systemNoiseCrossState

        Affect2State <- Affect2State + dAffect2 * 1/Nsteps +
          G * systemNoiseState2 + Gcross * systemNoiseCrossState
      }
    }
  }
}

```

```

        }
    }

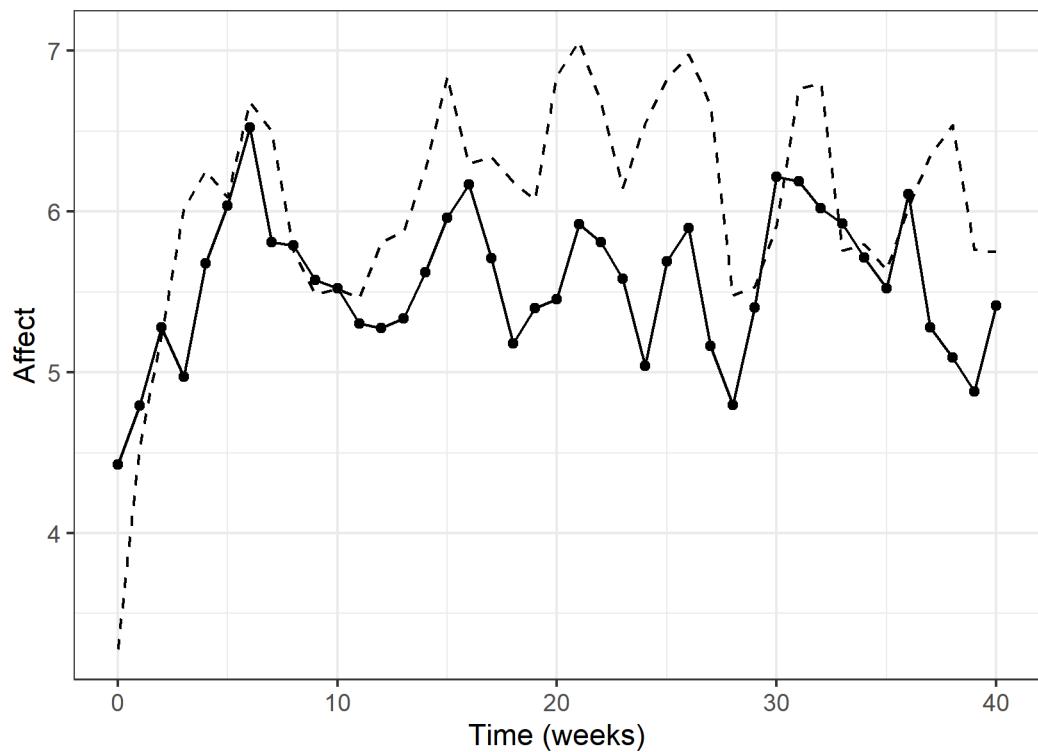
    data$Affect1[row] <- Affect1State #input affect data
    data$Affect2[row] <- Affect2State #input affect data
    data$Time[row] <- times[obsi] #input time data
    data$Comm[row] <- comm_val
    data$Subject[row] <- subi #input subject data
    data$TimeTogetherZ[row] <- TimeTogetherZ[subi] #input time together data
    data$const[row] <- 1 # add constant for ctsem spec
}

}

data$Affect1 <- data$Affect1 + rnorm(n=nrow(data), mean = 0, sd = .05) #add measurement error
data$Affect2 <- data$Affect2 + rnorm(n=nrow(data), mean = 0, sd = .05) #add measurement error

plot_trajectory(data[data$Subject==1], y_var = "Affect1", y2_var = "Affect2")

```



Model Fitting

To specify a model with a time-dependent moderator in `ctsem` we need to take four new steps. First, we need to specify the name of our time-dependent predictor using `TDPredNames = c("Comm")` where `Comm` is the name we give to our moderator. Second, we need to define the main effect of our predictor using the `TDPREDEFFECT` argument. By default this is set to freely

estimate a main effect for a time-dependent predictor. However, in our case we only want an interaction effect so we are going to set this to 0, `TDPREDEFFECT = 0`. Third, we need to modify our drift matrix. Here we are going to specify the interaction between our cross-effect state dependence terms and our time-dependent moderator, by specifying that our cross-effect state dependence is a function of an intercept (capturing a baseline cross-effect, i.e., when our moderator is 0) and an interaction term capturing the current level of our cross-effect which is moderated (multiplied) by the value of our moderator at a given time point, e.g., our cross-effect becomes `cross12int + cross12level*Comm`. To specify our cross-effect as a function of multiple elements, we need to use the `PARS` argument to tell `ctsem` which text strings are parameters to be estimated, and in this case, also that they are randomly varying across subjects. This gives us entries for `PARS` such as `cross12int||TRUE` – note that `cross12int` is referenced in the `DRIFT` matrix as part of a more complex equation defining the specific matrix element.

```
# Fit continuous time structural equation model
ct_model <- ctModel( #define the ctsem model
  TIpredNames = c('TimeTogetherZ'), #names of time independent predictors in dataset
  manifestNames = c("Affect1", "Affect2"), #names of observed variables in dataset
  latentNames = c("Affect1", "Affect2"), #names of latent processes
  TDpredNames = c('Comm'), #names of time dependent predictors in dataset
  time = 'Time', #name of time column in dataset
  id = 'Subject', #name of subject column in dataset
  type='ct', #use continuous time / differential equation model (dt for discrete-time)
  MANIFESTVAR = c(
    'residualSD1', 0,
    0, 'residualSD2'), #sd of the residual / measurement error
  LAMBDA = diag(1,2), #relating latent process to observed variables
  MANIFESTMEANS=0, #no measurement intercept (1 observed variable relates directly to latent)
  CINT=c('B1||TRUE', 'B2||TRUE'), #continuous intercept with random effects
  TOMEANS=c('initialAffect1||TRUE', 'initialAffect2||TRUE'), #initial affect with random effects
  TDPREDEFFECT = 0,
  DRIFT = c( #auto effects on diagonal, cross effects on off diagonal
    'auto1', '(cross12int + cross12level*Comm)', #auto effect for subj 1 and cross-effect from 2 to
    '(cross21int + cross21level*Comm)', 'auto2'), #cross-effect from 1 to 2 and auto effect for subj 2
  DIFFUSION = c(
    'systemNoise1', 0, #system noise person 1, 0 in upper triangle (correlation needs 1 par)
    'systemNoiseCross', 'systemNoise2'), #correlation in system noise, system noise person 2
  PARS = c('cross12int||TRUE', 'cross21int||TRUE',
    'cross12level', 'cross21level'))
ctModelLatex(ct_model) #generate LaTeX representation of the model
```

Nonlinear Change with a Transient External Shock (Continuous Time)

The last element we will consider in this tutorial is how to explicitly model an external influence at a specific time point with a specific form. So let's return to our example of Jill to make things more concrete.

Let's say Jill is going through therapy and improving at a rate given by her dynamic system

parameters that we used so far. What happens if she encounters a traumatic event in the midst of her 21 weeks of therapy? Such an event could have a pronounced effect on Jill's affect dynamics, which a realistic model of her growth needs to consider. We can incorporate this into our dynamic systems model using an input effect that captures the influence of an external intervention on the system dynamics.

Simulation

To add an input effect to our data generating model, we will make use of an `ifelse` statement within our data generating loop. We need to take two steps to make this work. First, we need to define the precise time point where we want our input to occur. In our case, we choose to intervene after half of our time window has passed `inputTime <- times[ceiling(length(times)/2)]`. Second, for each iteration in our loop, we evaluate the time point at each observation to see if it matches the time of the intervention we have specified (i.e. `inputTime`). When the answer becomes yes, then we add our input effect to the model that generates the rate of change in affect during that observation. This is done by adding `M` to the generated rate of change (`dAffect + M`) during the given observation, where `M` is the input effect coefficient determining the effect size of our intervention.

To save the input variable in our data frame, we add a column called `input` where the value is 1 when `inputTime` matches `Time`, and 0 otherwise.

```
# Generate data for multiple subjects with individual differences
NSubjects <- 20
times <- seq(from=0, to=20, by=1) #generate sequence of time points when subjects are measured
Nobs <- length(times) #number of observations per subject
initialAffect <- rnorm(n = NSubjects, mean = 5, sd = 2)
A <- -.1 #continuous time state dependence
B <- 1 #continuous intercept
G <- .2 #system noise coefficient
M <- -2 #input effect coefficient
inputTime <- times[ceiling(length(times)/2)] #intervention after ~ 1/2 of time window passed

#create empty data.frame to fill step by step
data <- data.frame(Subject= rep(NA,NSubjects*Nobs),
                    Time = rep(NA,NSubjects*Nobs),
                    Affect = rep(NA,NSubjects*Nobs))

Nsteps <- 100 #number of steps in time to compute between each observation (increased precision)

row <- 0 #initialize row counter, to track which row of the data.frame we are on
for(subi in 1:NSubjects){
  for(obsi in 1:Nobs){ #for each observation of a subject
    row <- row + 1
    if(obsi==1) AffectState <- initialAffect[subi] #if first time point, set to initial affect
    if(obsi>1){ #else compute new affect state by taking a sequence of small steps in time
      for(stepi in 1:Nsteps){ #take Nsteps in time between each observation
        dAffect <- A*AffectState + B #compute deterministic slope of affect at earlier time point
        if(times[obsi]==inputTime) dAffect <- dAffect + M #add input effect if intervention time
        AffectState <- AffectState + dAffect * 1/Nsteps + #update state using slope and time step
        G * rnorm(n=1, mean=0, sd=sqrt(1/Nsteps)) #and add system noise
      }
    }
  }
}
```

```

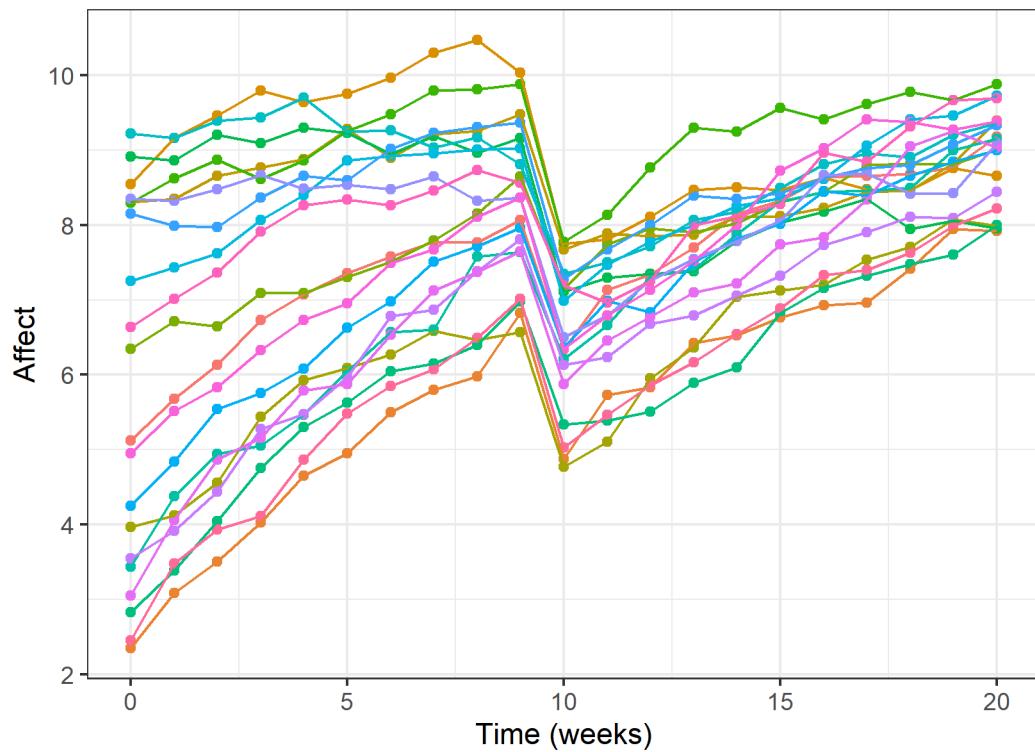
    }
}

data$Affect[row] <- AffectState #input affect data
data$Time[row] <- times[obsi] #input time data
data$Subject[row] <- subi #input subject data
data$Input <- ifelse(data$Time == inputTime, 1, 0) #input effect data
}

data$Affect <- data$Affect + rnorm(n=nrow(data), mean = 0, sd = .05) #add measurement error

plot_trajectory(data, color_var = "as.factor(Subject)")

```



Now we have a depiction of affect dynamics with an input effect, which decreases affect by 2 units at the moment it is applied to our system.

Model Fitting

To specify a ctsem model with an intervention effect, we simply need to specify a time-dependent predictor `TDpredNames = Input`. This adds a within-person covariate that has the same effect on each person in our dataset.

```

# Fit continuous time structural equation model
ct_model <- ctModel( #define the ctsem model
  manifestNames = "Affect", #names of observed variables in dataset
  latentNames = "Affect", #names of latent processes
  TDpredNames = 'Input', #names of time dependent predictors in dataset
  time = 'Time', #name of time column in dataset
  id = 'Subject', #name of subject column in dataset
  type='ct', #use continuous time / differential equation model (dt for discrete-time)
  MANIFESTVAR = 'residualSD', #sd of the residual / measurement error
  LAMBDA = matrix(1,nrow=1,ncol=1), #relating latent process to observed variables
  MANIFESTMEANS=0, #no measurement intercept (1 observed variable relates directly to latent)
  CINT='B||FALSE', #continuous intercept with *no* random effects
  TOMEANS='initialAffect||TRUE', #initial affect with random effects
  DRIFT = 'stateDependence',
  DIFFUSION = 'systemNoise')

ctModelLatex(ct_model) #generate LaTeX representation of the model

```

Subject parameter distribution:

$$\underbrace{[\text{initialAffect}_i]}_{\phi(i)} \sim \text{tform}\{\text{N}([\text{raw_initialAffect}], [\text{rawPCov_1_1}])\}$$

Initial latent state:

$$\underbrace{[\text{Affect}](t_0)}_{\eta(t_0)} \sim \text{N}\left(\underbrace{[\text{initialAffect}]}_{\text{TOMEANS}}, \underbrace{UcorSDtoCov\{[1e-06]\}}_{\underbrace{\text{Q}^*}_{\text{TOVAR}} t_0}\right)$$

Deterministic change:

$$\frac{d[\text{Affect}](t)}{d\eta(t)} = \left(\underbrace{[\text{stateDependence}]}_{\text{DRIFT}} \underbrace{[\text{Affect}](t)}_{\eta(t)} + \underbrace{[\text{B}]}_{\text{CINT}} + \underbrace{[\text{td_Affect_Input}]}_{\text{TDPREDEFECT}} \underbrace{[\text{Input}]}_{x(t)} \right) dt +$$

Random change:

$$\underbrace{UcorSDtoChol\{[\text{systemNoise}]\}}_{\text{G}} \underbrace{d[W_1](t)}_{dW(t)}$$

Observations:

$$\underbrace{[\text{Affect}](t)}_{Y(t)} = \underbrace{[1]}_{\Lambda} \underbrace{[\text{Affect}](t)}_{\eta(t)} + \underbrace{[0]}_{\tau} + \underbrace{\text{MANIFESTMEANS}}_{+}$$

Observation noise:

$$\underbrace{[\text{residualSD}]}_{\Theta} \underbrace{[\epsilon_1](t)}_{\epsilon(t)}$$

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

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

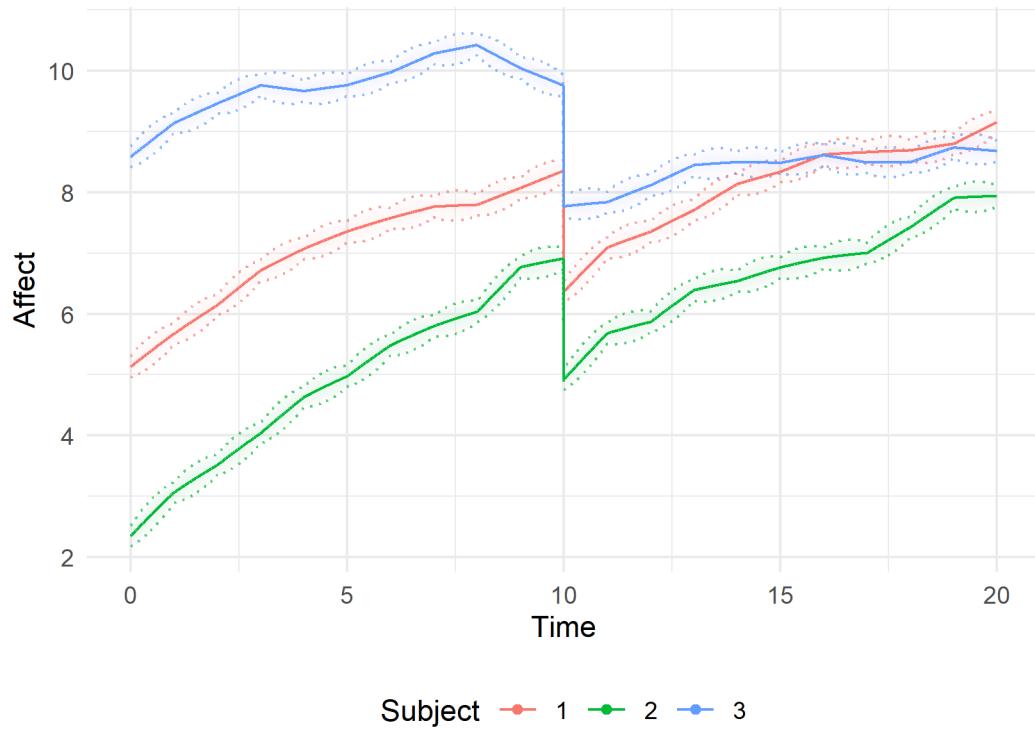
Note: *UcorSDtoChol* converts lower tri matrix of standard deviations and unconstrained correlations to Cholesky factor, *UcorSDtoCov* = transposed cross product of *UcorSDtoChol*, to give covariance, See Driver & Voelkle (2018) p11.
 Individual specific notation (subscript i) only shown for subject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

Looking at the model, note that we now have a time dependent predictor effect – this is the input effect `td_Affect_Input`.

```
ct_fit <- ctStanFit(datalong = data, ctstanmodel = ct_model) #fit the model to our data
```

After fitting, we can plot the ‘smoothed’ estimates of either the latent state or measurement values, which is conditional on all the data available – not just past data as in the default predictions when using `ctKalman`. Here we plot the smoothed measurement values for the first three subjects in our dataset, which shows the immediate drop in affect at the time of the intervention, followed by a gradual recovery over time.

```
ctKalman(ct_fit, plot=TRUE, subjects=1:3,
  kalmanvec='ysmooth', timestep = .01)
```



Different Types of Inputs

The model above implies that the input effect is immediate and has a lasting effect on the system, as we can see in the plot where the level takes time to recover. But there is also the assumption that the effects transmit through time with the same dynamics as the rest of the system. This is probably the simplest approach to thinking about such effects, but in reality, the effects of an intervention may be more complex, and may depend on the individual, the context, may induce other changes in the system, and may persist or transfer through the system in different ways to the typical system behavior. For example, while the regular ups and downs of life may dissipate relatively quickly, a traumatic event may have direct effects that last over a long period and may require more consideration.

Nonlinear Change with a Persistent External Shock (Continuous Time)

For our example, we can model an intervention that has a persistent effect on all future instances of our system. For more examples of intervention effects and how to specify them in `ctsem` we refer the reader to ([Driver and Voelkle 2018b](#)).

The simplest way to add a persistent external shock would be to just use a dummy variable that applies a 0 to all observations prior to the intervention and a 1 to all observations post (and including) the time of the intervention. However, this would lead to a spike in the rate of change in affect during our observation time, followed by a gradual decay in the rate of change back to baseline in the time in-between observations. This can cause problems when used with specific study designs (e.g. unequal time intervals). Thus, a solution that is more generally appropriate is desirable.

To do this, we need to specify a latent (intervention) process which is impacted by our observed intervention variable. In turn, the latent intervention process will directly impact our affect process *during and in-between our observations*. This latent intervention process, can have its own autoregressive parameter which allows us to specify an accelerating, dissipating, or stable effect across continuous time. That is, if the autoregressive parameter is set to 0, our latent intervention process will remain at a stable value and exert a stable effect on a person's rate of change over time. If the autoregressive parameter is set to negative value (e.g. -0.1), then the effect of the intervention will dissipate over time. If it is set to a value greater than 0, the effect of our intervention will accumulate over time (e.g. 0.1). In our example, we will demonstrate an intervention with a persistent effect over time that is administered during the halfway mark of our observed timeline.

Simulation

In our data generating block, we now include an input variable `M`, which directly influences the state of our latent intervention process `InputState`. To understand how this works let us begin from the equation that defines the `InputState` at each time step: `InputState <- InputState + dInput * 1/Nsteps`. So the `InputState` at a given time step, is given by the `InputState` at the prior time step, plus the rate of change `dInput * 1/Nsteps`.

The rate of change in our latent intervention process `dInput` is given by the previous value of `InputState` multiplied by an autoregressive coefficient, `dInput <- AM*InputState`. Because the autoregressive coefficient is 0 in the case of a persistent input effect, the rate of change is also 0, until the observed intervention occurs.

At the time of the intervention, week 50, the input effect `M` is added to the equation for the rate of change `dInput <- AM*InputState + M`. This raises the `InputState` to the value of `M`.

After week 50, the rate of change becomes zero again. Because the autoregressive parameter is zero. Thus, since `InputState` equals its own prior state + (zero) change, the state of the latent intervention process `InputState` will not change; reflecting a persistent input process.

Finally, to model the impact of our latent intervention process on our affect process, we make the rate of change in affect dependent on the state of the intervention process at the current moment `dAffect <- A*AffectState + B + InputState`.

```
# Generate data for multiple subjects with individual differences
NSubjects <- 20
times <- seq(from=0, to=100, by=1) #generate sequence of time points when subjects are measured
Nobs <- length(times) #number of observations per subject
initialAffect <- rnorm(n = NSubjects, mean = 5, sd = 2)
A <- -.1 #continuous time state dependence
B <- 1 #continuous intercept
G <- .2 #system noise coefficient
M <- -0.4 #input variable
AM <- 0 #latent input process autoregression
inputTime <- times[ceiling(length(times)/2)] #intervention after ~ 1/2 of time window passed

#create empty data.frame to fill step by step
data <- data.frame(Subject= rep(NA,NSubjects*Nobs),
                    Time = rep(NA,NSubjects*Nobs),
                    Affect = rep(NA,NSubjects*Nobs))

Nsteps <- 100 #number of steps in time to compute between each observation (increased precision)

row <- 0 #initialize row counter, to track which row of the data.frame we are on
for(subi in 1:NSubjects){
  for(obsi in 1:Nobs){ #for each observation of a subject
    row <- row + 1
    if(obsi==1){
      AffectState <- initialAffect[subi] #if first time point, set to initial affect
      InputState <- 0 # initialize latent input process
    }
    if(obsi>1){ #else compute new affect state by taking a sequence of small steps in time
      for(stepi in 1:Nsteps){ #take Nsteps in time between each observation

        # Compute intervention effect
        dInput <- AM*InputState # deterministic slope of our intervention at earlier time point
        if(times[obsi]==inputTime) dInput <- dInput + M # add input effect if intervention time

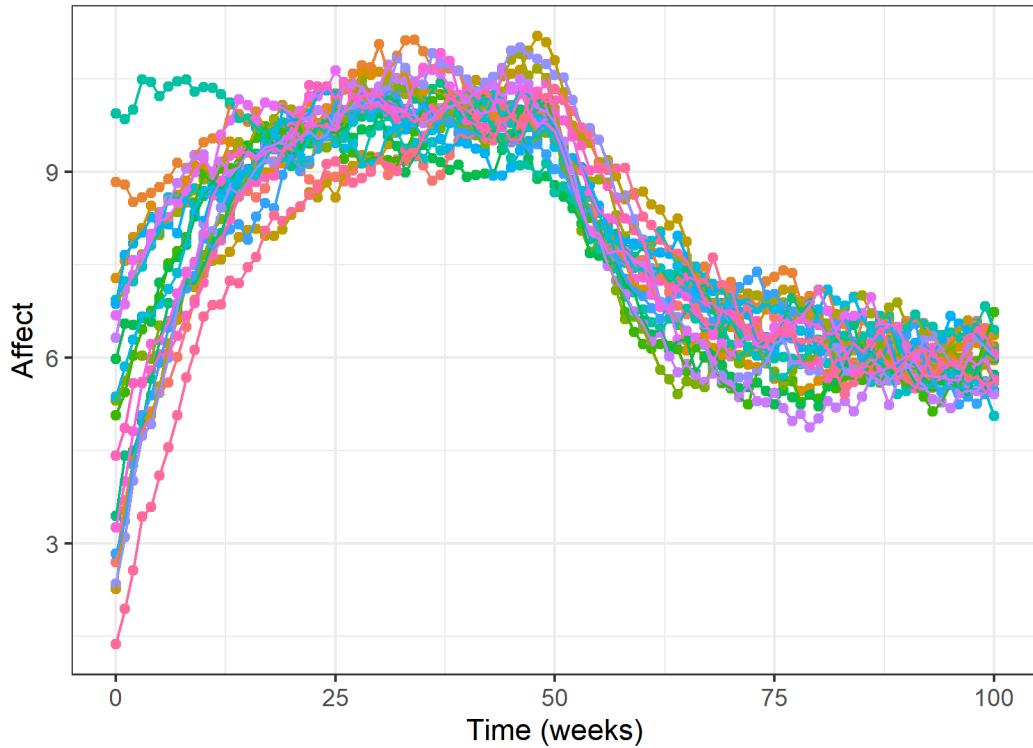
        InputState <- InputState + dInput * 1/Nsteps # update state using slope and time step

        # Compute Affect state
        dAffect <- A*AffectState + B + InputState # deterministic slope at earlier time point
        AffectState <- AffectState + dAffect * 1/Nsteps + # update state using slope and time step
        G * rnorm(n=1, mean=0, sd=sqrt(1/Nsteps)) # and add system noise
      }
    }
    data$Affect[row] <- AffectState #input affect data
    data$Time[row] <- times[obsi] #input time data
    data$Subject[row] <- subi #input subject data
    data$Input <- ifelse(data$Time == inputTime, 1, 0) #input effect data
    data$InputState[row] <- InputState #input latent intervention process data
  }
}
```

```

data$Affect <- data$Affect + rnorm(n=nrow(data), mean = 0, sd = .05) #add measurement error
plot_trajectory(data, color_var = "as.factor(Subject)")

```



Model & Fitting

We want to specify a latent intervention process that only captures the effect of our observed intervention variable, without any extra noise or baseline effects. To do this, we set all the extra influences on the latent intervention process, that do not come directly from the observed intervention variable, to 0. Thus, we set the random fluctuations (DIFFUSION) to zero for our latent intervention process. We also set its starting value to 0, by fixing the TOMEANS and TOVAR to zero for our latent intervention process. Finally, we fix its continuous intercept to zero CINT since there is no constant growth in the latent intervention process.

Next, we set the cross-effect of our latent process to 1 using our DRIFT matrix (adding a 1 in row 1, column 2 of the matrix). This, specifies a direct effect of the current state of the latent intervention process on the rate of change in our affect process.

Lastly, we set the effect of the observed intervention variable on our latent affect process to zero, and estimate its direct effect on our latent intervention process (TDPREDEFFECT =

`matrix(c(0,"LatentInput"), nrow=2, ncol=1)).` Thus, our observed intervention directly impact the latent intervention process, which in turn directly impacts our latent affect process.

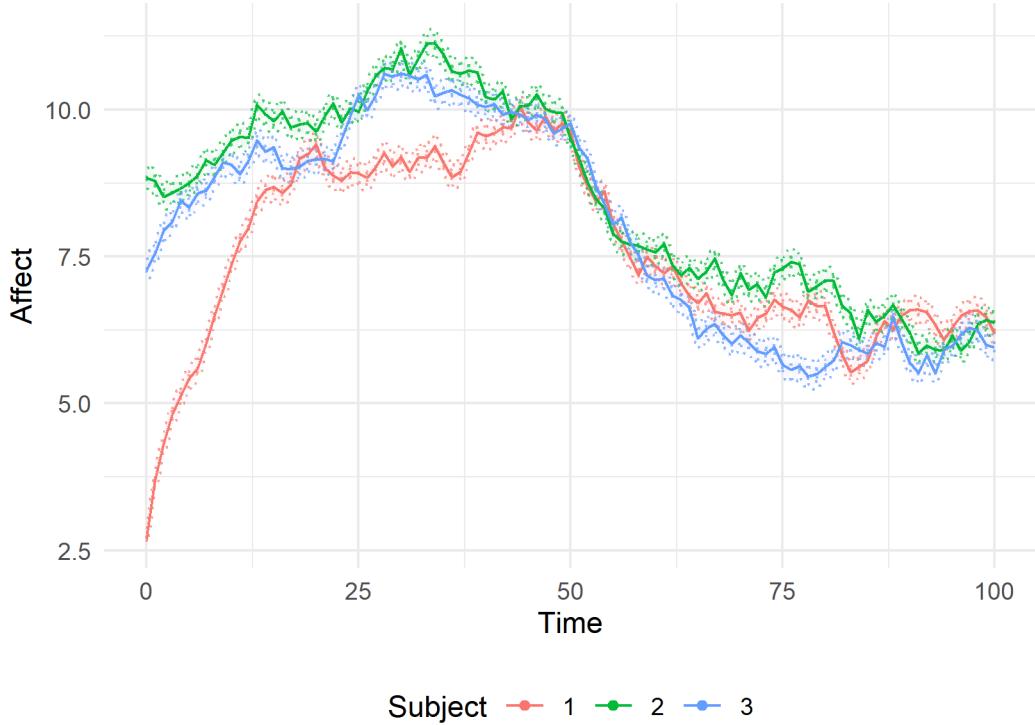
```
# Fit continuous time structural equation model
ct_model <- ctModel( #define the ctsem model
  manifestNames = "Affect", #names of observed variables in dataset
  latentNames = c("Affect","LatentInput"), #names of latent processes
  TDpredNames = 'Input', #names of time dependent predictors in dataset
  time = 'Time', #name of time column in dataset
  id = 'Subject', #name of subject column in dataset
  type='ct', #use continuous time / differential equation model (dt for discrete-time)
  MANIFESTVAR = 'residualSD', #sd of the residual / measurement error
  LAMBDA = matrix(1,nrow=1,ncol=2), #relating latent process to observed variables
  MANIFESTMEANS=0, #no measurement intercept (1 observed variable relates directly to latent)
  CINT=c('B||FALSE', #continuous intercept with *no* random effects for Affect,
    "0||FALSE"), # and no CINT for latent input
  TDPREFFECT = matrix(c(0,"LatentInput"), nrow=2, ncol=1),
  TOMEANS=c("initialAffect||TRUE", #initial affect with random effects,
    "0||FALSE"), # and latent input process starts at 0
  TOVAR=matrix(c("TOVARAffect",0,0,0), nrow=2, ncol=2),
  DRIFT = matrix(c('stateDependence', 1,
    0, 0), nrow = 2, ncol = 2, byrow = TRUE),
  DIFFUSION = matrix(c('systemNoise',0,
    0, 0), nrow = 2, ncol = 2, byrow = TRUE))

ctModelLatex(ct_model) #generate LaTeX representation of the model
```

```
ct_fit <- ctStanFit(datalong = data, ctstanmodel = ct_model) #fit the model to our data
```

Using the same approach as earlier, we can plot the smoothed estimates of the observed variable states (latent states if desired using ‘etasmooth’ instead of ‘ysmooth’) for the first three subjects in our dataset, which shows the drop in affect at the time of the intervention, followed by a gradual recovery over time. Note here that the drop is not immediate, but persists in comparison to the earlier example.

```
ctKalman(ct_fit, plot=TRUE, subjects=1:3,
  kalmanvec='ysmooth', timestep = .01)
```



Model Evaluation and Comparison

To end this tutorial, we are going to investigate how to evaluate and compare different model structures when we do not know the structure of our observed data. In empirical settings, this can be a very difficult and fascinating problem that requires deep inquiry. To get people started on their journey of model evaluation, we will showcase some tools that can help determine whether a model provides a good representation of the data and whether it is better than any competing explanations (models) we may have. We will begin from a familiar starting place, and then enrich our model evaluation by introducing new tools.

The most common form of model evaluation happens using a very simple method known as the R^2 ([Nakagawa and Schielzeth 2013](#)). The R^2 tells us how much of the total variance in our data is explained by our model by dividing the residual (unexplained) variance ($V\hat{y}$) by the total variance(Vy). Because the total variance is not necessarily constant across individuals or time, this is computed using standardized residuals, and reported (as standardized residual variance) at the top of the ctsem summary output.

So let's imagine that after all the model building we have done, we still think that the best representation of an affect process is a linear model in continuous time. If we have data from 20 patients over 21 weeks, we can use the R^2 estimate to check whether our model explains a

substantial amount of variance in our data (what counts as substantial is a topic for another paper). We will generate the data and pretend it is observed, because knowing the answers simplifies the process of understanding model evaluation. So we will generate data using from a continuous-time model with state dependency and random fluctuations.

Generating Data

To generate our data we will recycle our code block from earlier (see ‘Simulation: Nonlinear Change with Random Fluctuations (Continuous Time)’).

```
# Generate data for multiple subjects with individual differences
NSubjects <- 20
times <- seq(from=0, to=20, by=1) #generate sequence of time points when subjects are measured
Nobs <- length(times) #number of observations per subject
initialAffect <- rnorm(n = NSubjects, mean = 5, sd = 2)
A <- -.1 #continuous time state dependence
B <- 1 #continuous intercept
G <- 0.2 #system noise coefficient

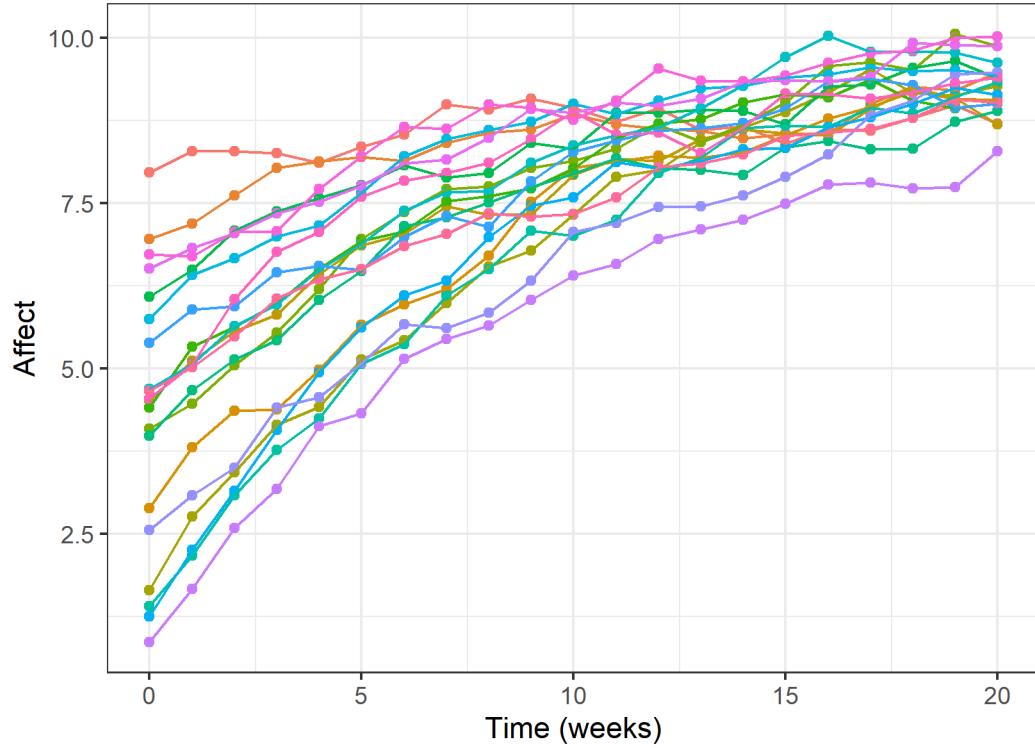
#create empty data.frame to fill step by step
data <- data.frame(Subject= rep(NA,NSubjects*Nobs),
Time = rep(NA,NSubjects*Nobs),
Affect = rep(NA,NSubjects*Nobs))

Nsteps <- 100 #number of steps in time to compute between each observation (increased precision)

row <- 0 #initialize row counter, to track which row of the data.frame we are on
for(subi in 1:NSubjects){
  for(obsi in 1:Nobs){ #for each observation of a subject
    row <- row + 1
    if(obsi==1) AffectState <- initialAffect[subi] #if first time point, set to initial affect
    if(obsi>1){ #else compute new affect state by taking a sequence of small steps in time
      for(stepi in 1:Nsteps){ #take Nsteps in time between each observation
        dAffect <- A*AffectState + B #compute deterministic slope of affect at earlier time point
        AffectState <- AffectState + dAffect * 1/Nsteps + #update state using slope and time step
          G * rnorm(n=1, mean=0, sd=sqrt(1/Nsteps)) #and add system noise
      }
    }
    data$Affect[row] <- AffectState #input affect data
    data$Time[row] <- times[obsi] #input time data
    data$Subject[row] <- subi #input subject data
  }
}

data$Affect <- data$Affect + rnorm(n=nrow(data), mean = 0, sd = .05) #add measurement error

plot_trajectory(data, color_var = "as.factor(Subject)")
```



Fitting the wrong model: A linear model in continuous time

We can now specify and fit our linear continuous-time model, with no state-dependency and no system noise.

```
# Fit continuous time structural equation model
ct_model <- ctModel( #define the ctsem model
  # Specify features of the data
  manifestNames = "Affect", #names of observed variables in dataset
  latentNames = "Affect", #names of latent processes
  time = 'Time', #name of time column in dataset
  id = 'Subject', #name of subject column in dataset
  type='ct', #use continuous time / differential equation model (dt for discrete-time)
  # Specify features of the model
  MANIFESTVAR = 'residualSD', #sd of the residual / measurement error
  LAMBDA = matrix(1,nrow=1,ncol=1), #relating latent process to observed variables
  MANIFESTMEANS=0, #no measurement intercept (1 observed variable relates directly to latent)
  CINT='B|!FALSE', #continuous intercept with *no* random effects
  TOMEANS='initialAffect||TRUE', #initial affect with random effects
  DRIFT = 0,
  DIFFUSION = 0)

ctModelLatex(ct_model)
```

```
ct_fit <- ctStanFit(datalong = data, ctstanmodel = ct_model) #fit the model to our data

s=summary(ct_fit) #store summary of the fit, some output disabled
print(s$ResidCovStd)
```

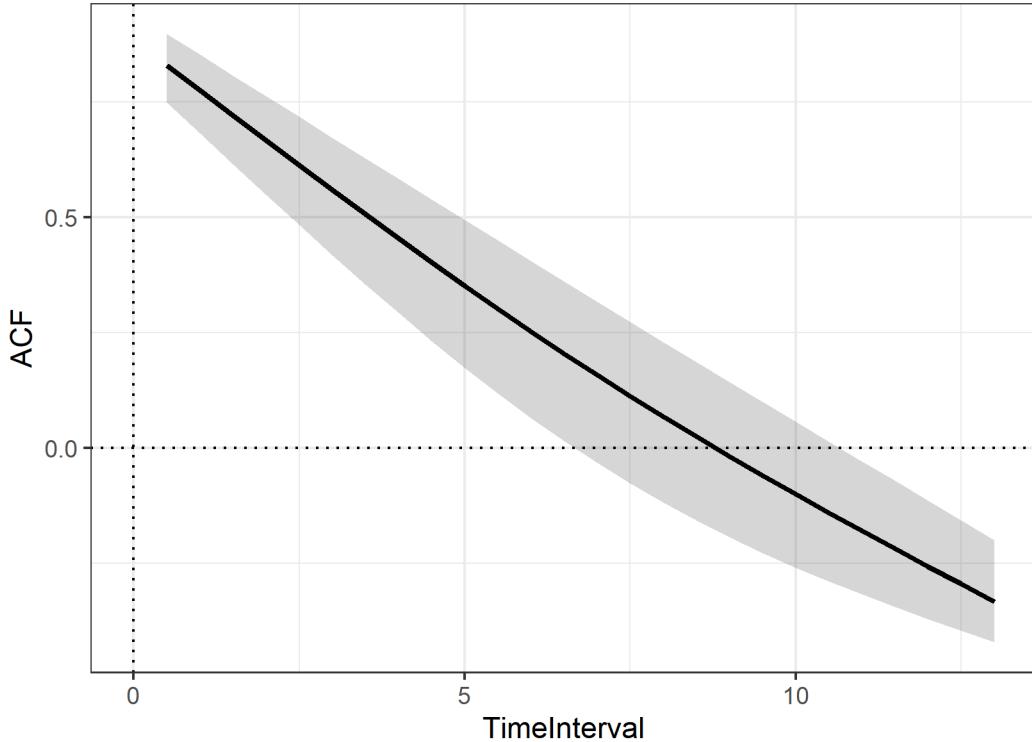
NULL

Now we can use the standardized residual variance estimate (reported at the top of summary under ‘ResidCovStd’, equivalent to $1 - R^2$) to see how much of the variance in our empirical data our model has accounted for. We can see that our model explains a very high amount of the variance in our observations. This is because it is almost the same model as the model that generated our data. At this point, in many empirical circumstances we would go home very satisfied with our model. Explaining almost 80 percent of the variance could mean that the remaining variance is simply noise.

But this can also be assessed. We already extracted the residuals from our model and we can check their autocorrelation. Autocorrelated residuals indicate that our model leaves some predictable structure unexplained ([Gelman, Meng, and Stern 1996](#)). While residuals that are not correlated are more likely to just be random noise.

We can create a plot of the autocorrelation between residuals to visualize whether our model is missing any systematic patterns in our data. To do this we use the `ctACFresiduals` function.

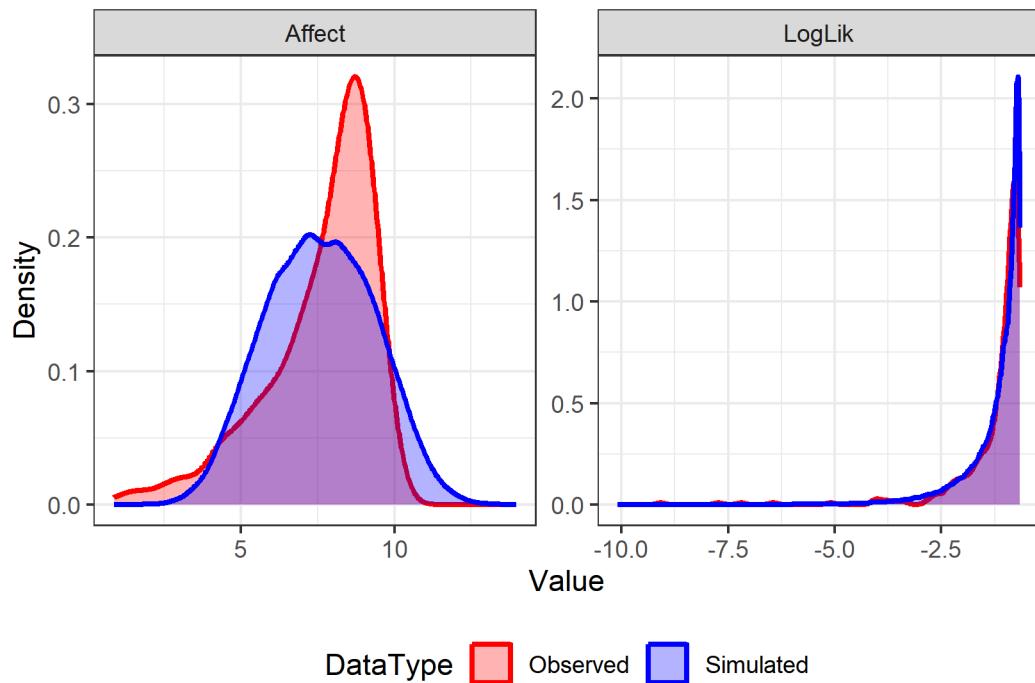
```
ctACFresiduals(ct_fit) #plot autocorrelation of residuals
```



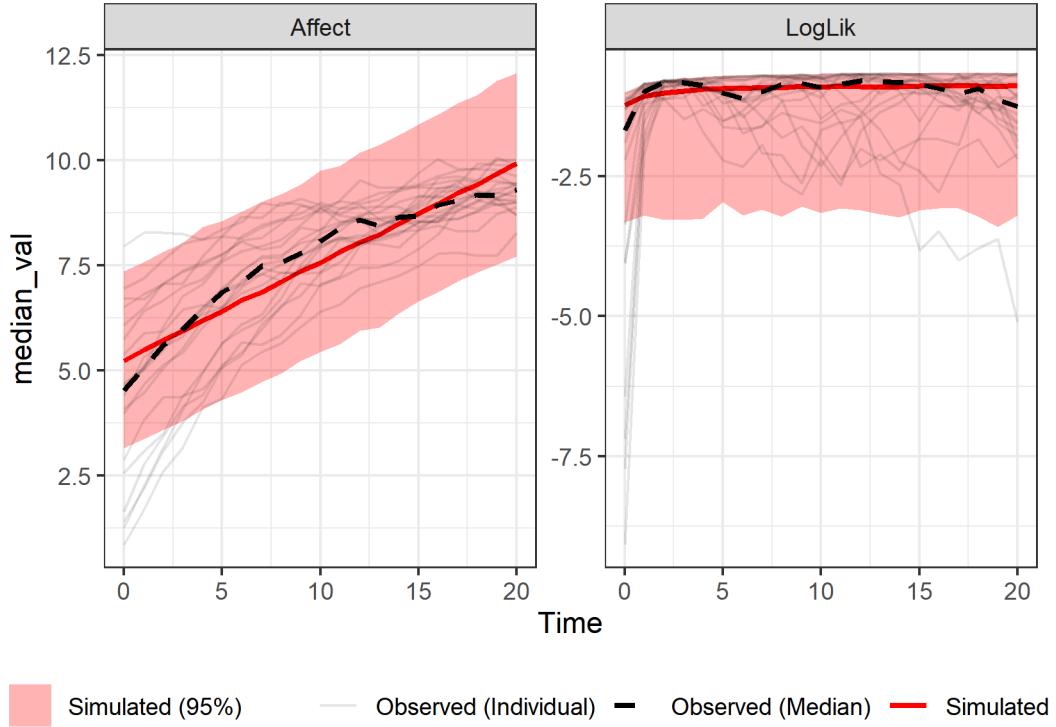
From the plot we can see that there is a high autocorrelation between residuals, about 0.6 between temporally adjacent residuals. So now we know two things: (1) Our model is doing a good job at capturing most of the systematic patterns in our data, (2) there is still room for improvement. The natural follow-up question is what should we improve? To get an idea of model misspecification we can generate data based on the parameter estimates of our model and plot those against our observed data.

```
fit <- ctStanGenerateFromFit(ct_fit, nsamples=200, #add generated data to the fit object
  fullposterior = F, cores = 2)
postpred <- ctPostPredPlots(ct_fit) #generate and store posterior predictive plots
print(postpred[[1]]) #print first plot -- overall density plots of likelihood / variables
```

Density of generated data and observed values



```
print(postpred[[2]]) #print second plot -- compares expectations / variation over time
```



By looking at the model implied expectations versus the empirical, it is quite clear that we are fitting a linear model to a nonlinear trajectory. Our model also seems to underestimate the between-subject variance at earlier time points while overestimating the variance at later observations. But in which way can we improve our model to better account for our observations?

By taking a closer look at the individual growth trajectories, we can see there seems to be a dependency between people's initial affect values and their rate of change in affect. That is, people who start with very high affect values have an almost flat growth curve, while people with a low initial level have a very steep growth curve which slowly plateaus as they reach higher values of affect. Thus, it seems that we should add a state-dependency parameter to our model.

Finally, since people largely have the same post-therapy baseline in affect (i.e. they plateau at similar values) we should keep the continuous intercept fixed across our population. So, let's go fit our adjusted model to our data.

Fitting a better model: Adding state-dependence in continuous time

```

# Fit continuous time structural equation model
ct_model <- ctModel( #define the ctsem model
  # Specify features of the data
  manifestNames = "Affect", #names of observed variables in dataset
  latentNames = "Affect", #names of latent processes
  time = 'Time', #name of time column in dataset
  id = 'Subject', #name of subject column in dataset
  type='ct', #use continuous time / differential equation model (dt for discrete-time)
  # Specify features of the model
  MANIFESTVAR = 'residualSD', #sd of the residual / measurement error
  LAMBDA = matrix(1,nrow=1,ncol=1), #relating latent process to observed variables
  MANIFESTMEANS=0, #no measurement intercept (1 observed variable relates directly to latent)
  CINT='B||FALSE', #continuous intercept with *no* random effects
  TOMEANS='initialAffect||TRUE', #initial affect with random effects
  DRIFT = 'stateDependence',
  DIFFUSION = 0)

ctModellLatex(ct_model)

ct_fit <- ctStanFit(datalong = data, ctstanmodel = ct_model) #fit the model to our data

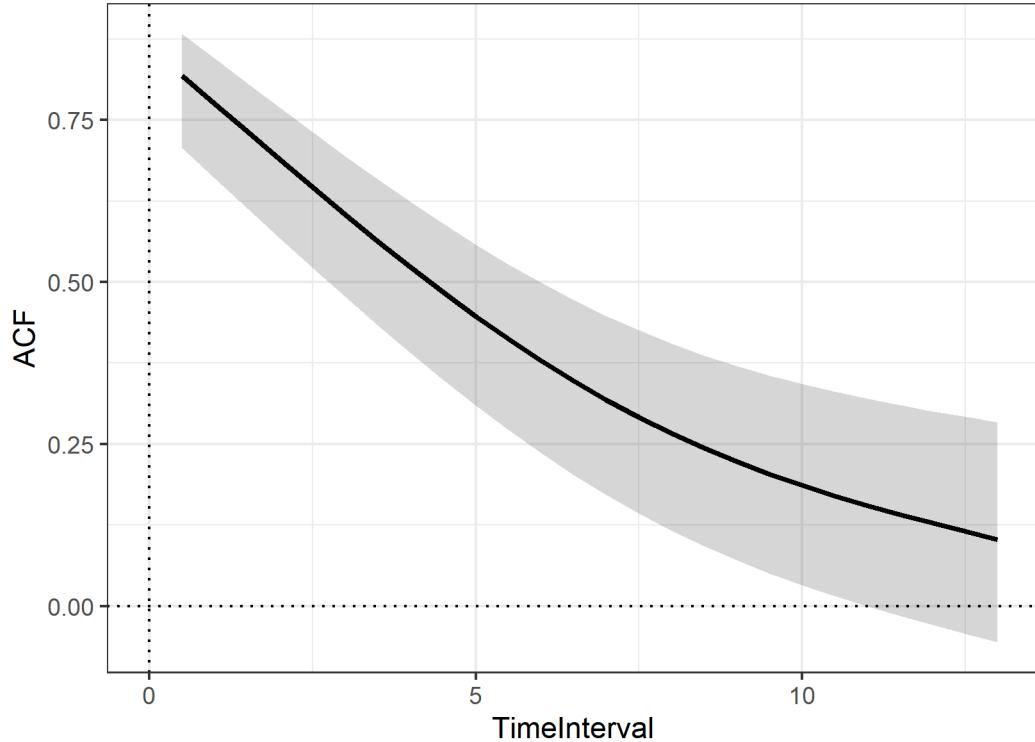
s=summary(ct_fit) #store summary of the fit, some output disabled
print(s$ResidCovStd)

```

NULL

Now we can again see if adding this state-dependence term has helped us explain more variance in our data by considering the R^2 – and indeed, we can see that we have around 10% residual variance, or 90% explained variation in our data – noticeably more than before. So now we can go view our autocorrelation plot to check if there is still information left in the residuals that our model could explain.

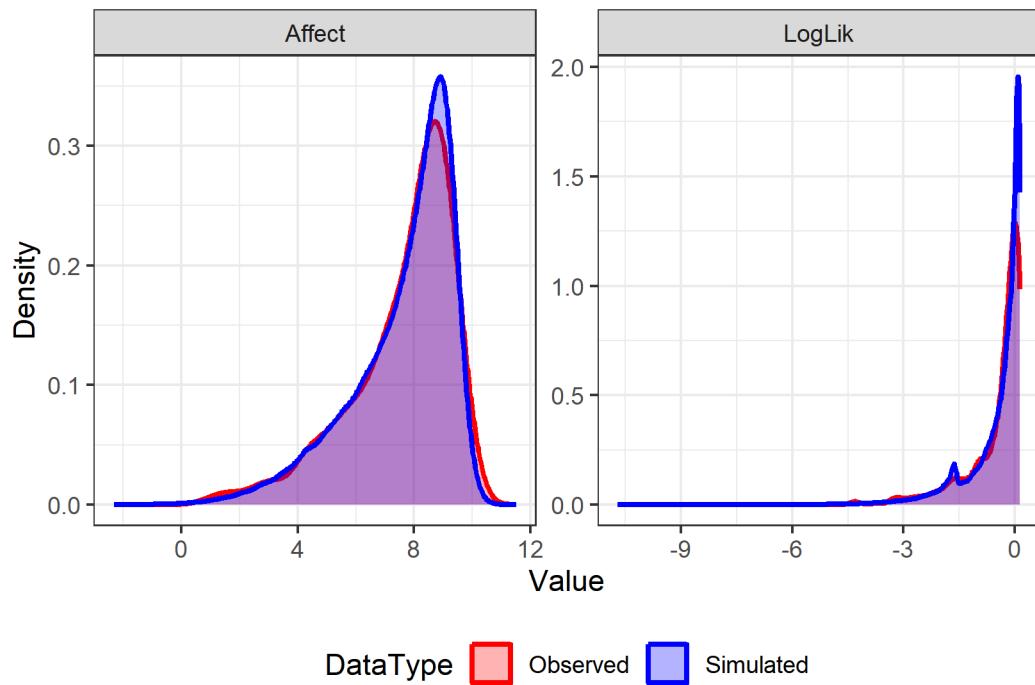
```
ctACFresiduals(ct_fit) #plot autocorrelation of residuals
```



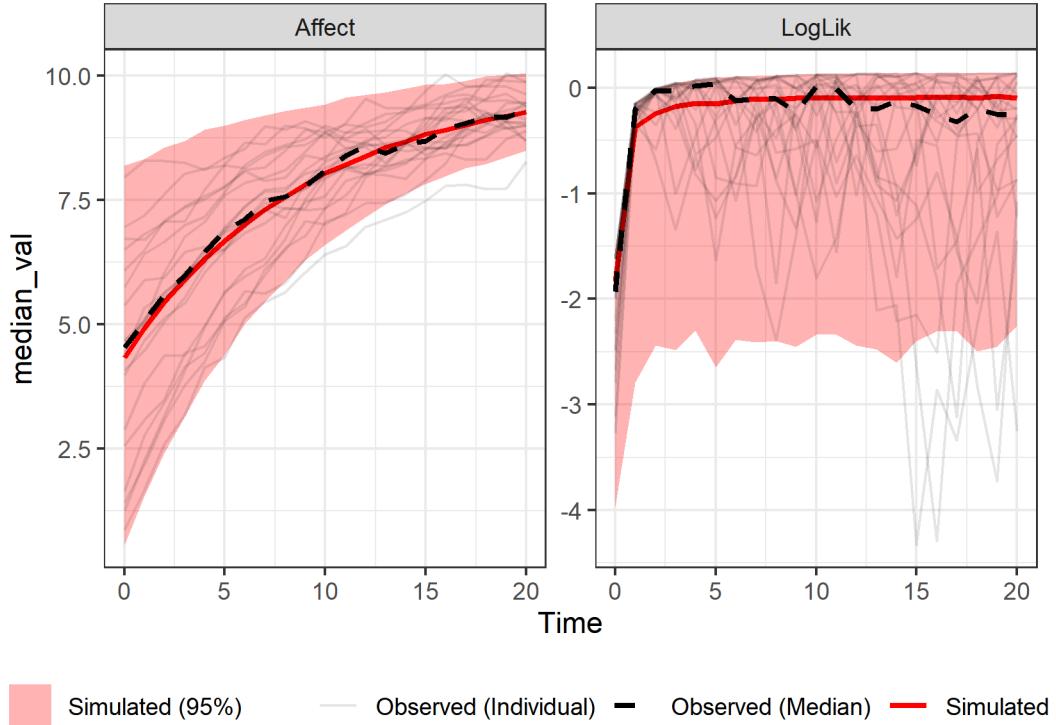
We can see that the autocorrelation has also dropped substantially, but there is still a moderate amount of autocorrelation in our residuals. Which is rightly telling us that there is more predictable variation that our model can leverage. So let's try to find out what it is by plotting the model generated data over our observed data.

```
fit <- ctStanGenerateFromFit(ct_fit, nsamples=200, #add generated data to the fit object
  fullposterior = F, cores = 2)
postpred <- ctPostPredPlots(ct_fit) #generate and store posterior predictive plots
print(postpred[[1]]) #print first plot -- overall density plots of likelihood / variables
```

Density of generated data and observed values



```
print(postpred[[2]]) #print second plot -- compares expectations / variation over time
```



This time we see that our model implied trajectory almost perfectly fits our observed trajectory. It is not so easy to determine what information we should add to our model. We can reason through domain knowledge (but mostly because we generated the data) that affect processes are often inherently noisy. That is the growth of affect is influenced by many random life events which can push its rate of growth up and down at each moment. The black lines in our plot also indicate that our observed trajectories are noisy. This could be measurement error or system noise. We can add a system noise term to find out.

Fitting the true model: Adding state-dependence and system noise in continuous time

```
# Fit continuous time structural equation model
ct_model <- ctModel( #define the ctsem model
  # Specify features of the data
  manifestNames = "Affect", #names of observed variables in dataset
  latentNames = "Affect", #names of latent processes
  time = 'Time', #name of time column in dataset
  id = 'Subject', #name of subject column in dataset
  type='ct', #use continuous time / differential equation model (dt for discrete-time)
  # Specify features of the model
  MANIFESTVAR = 'residualSD', #sd of the residual / measurement error
```

```

LAMBDA = matrix(1,nrow=1,ncol=1), #relating latent process to observed variables
MANIFESTMEANS=0, #no measurement intercept (1 observed variable relates directly to latent)
CINT='B||FALSE', #continuous intercept with *no* random effects
TOMEANS='initialAffect||TRUE', #initial affect with random effects
DRIFT = 'stateDependence',
DIFFUSION = 'systemNoise')

```

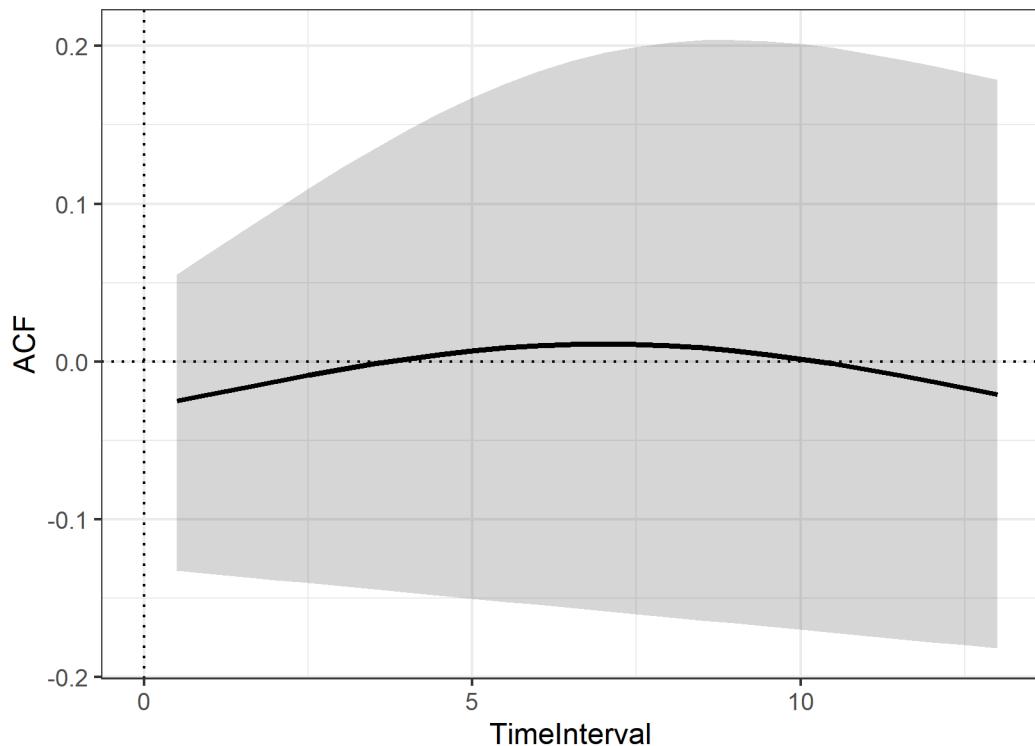
```
ctModelLatex(ct_model)
```

```
ct_fit <- ctStanFit(datalong = data, ctstanmodel = ct_model) #fit the model to our data
```

```
s=summary(ct_fit) #store summary of the fit, some output disabled
print(s$ResidCovStd)
```

NULL

```
ctACFresiduals(ct_fit) #plot autocorrelation of residuals
```

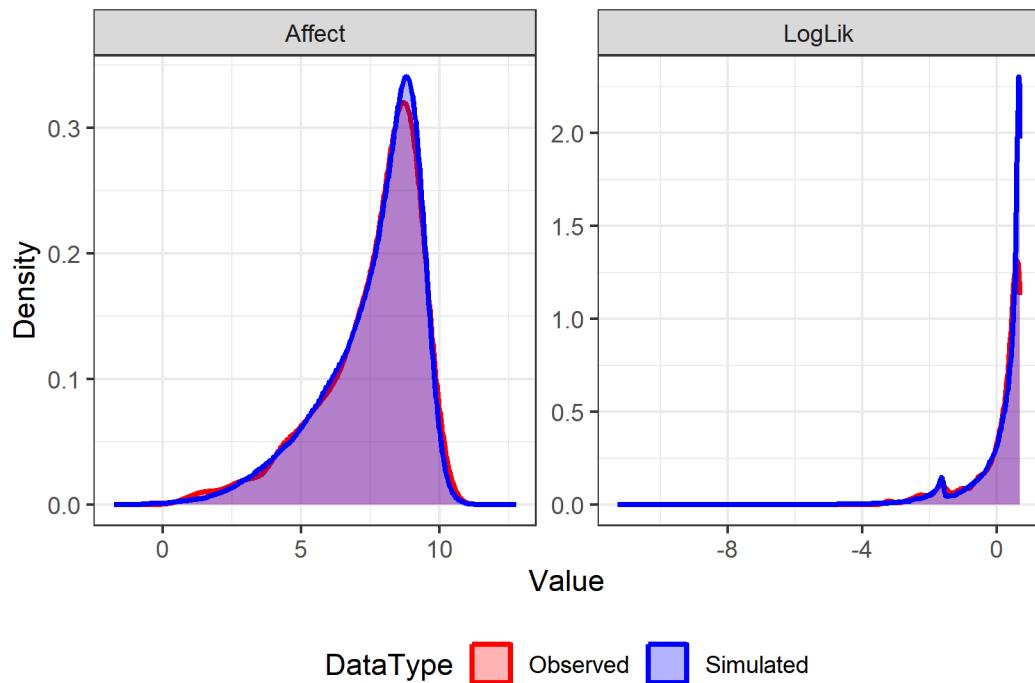


```

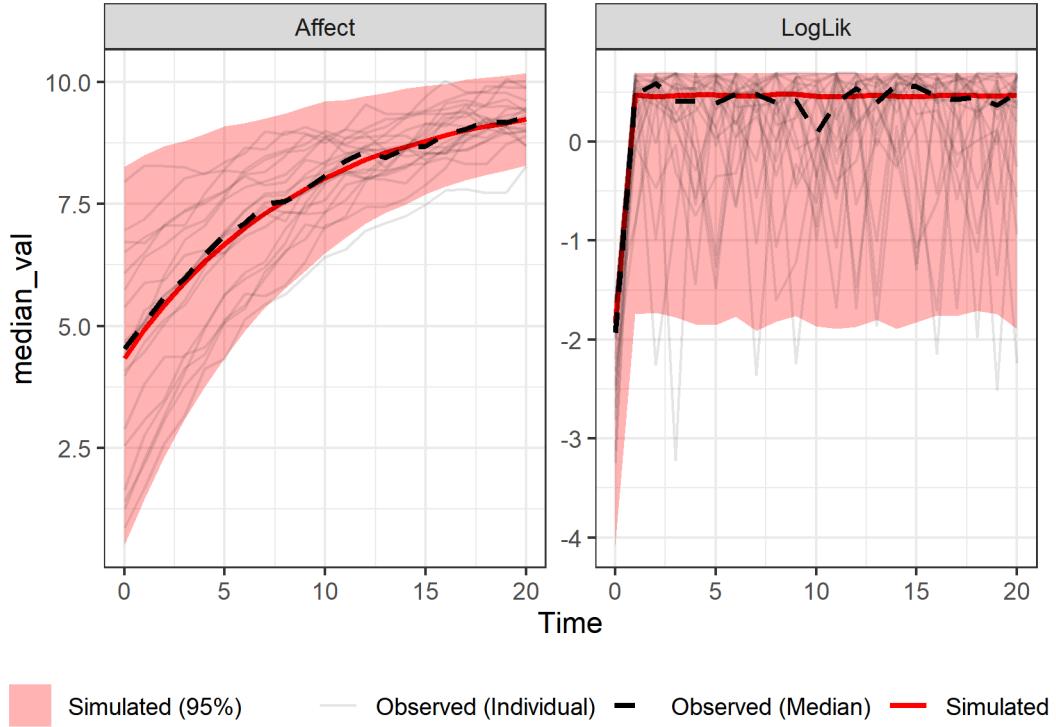
fit <- ctStanGenerateFromFit(ct_fit, nsamples=200, #add generated data to the fit object
  fullposterior = F, cores = 2)
postpred <- ctPostPredPlots(ct_fit) #generate and store posterior predictive plots
print(postpred[[1]]) #print first plot -- overall density plots of likelihood / variables

```

Density of generated data and observed values



```
print(postpred[[2]]) #print second plot -- compares expectations / variation over time
```



Overall, we can see that fitting the model to the data it generated leads to a greater R^2 , a near-zero autocorrelation between residuals, and a model-implied trajectory that maps as perfectly as we can wish for onto the observed trajectory.

In empirical contexts, this process is going to be more complicated and a data-driven method to iterative model improvement runs the risk of making model modifications that explain chance characteristics of the data. Thus, raising questions about generalizability to other samples and the population of interest. To mitigate such concerns, tools to circumvent overfitting can be used, such as iteratively building a model using a subsample of the data and then evaluating the model on unseen data (for more examples see Yarkoni and Westfall (2017)).

Further Works and Resources

Beyond the core ctsem workflow, several tutorials and reviews map the broader longitudinal modelling landscape and can help with the array of choices to be made in linking theory to models and data. For discrete-time foundations, a concise, practice-oriented overview of growth curve models addresses common pitfalls and implementation details ([Curran, Obeidat, and](#)

([Losardo 2010](#)). For intensive longitudinal data that blur boundaries between multilevel models, time series, and SEM, dynamic structural equation modelling (DSEM) provides an integrated framework with worked examples and diagnostics ([Asparouhov, Hamaker, and Muthén 2018](#)). Bridging discrete- and continuous-time perspectives, step-by-step introductions show how cross-lagged and autoregressive panel models can be translated into their continuous-time counterparts with stochastic differential equations and estimated within SEM ([Voelkle et al. 2012](#)), including guidance for irregular sampling and oscillatory processes ([Voelkle et al. 2012](#)). For researchers using cross-lagged panel models specifically, some criticisms clarify interpretational limits and motivates model choices that better align statistical parameters with substantive within-person processes ([Berry and Willoughby 2017; Hamaker, Kuiper, and Grasman 2015](#)).

For dynamic-systems modelling that formalizes change mechanisms directly as differential (or stochastic differential) equations, there are other accessible software and methodological resources. The `dynr` package paper provides a hands-on introduction to specifying linear/nonlinear, discrete/continuous-time models (including regime switching) with code recipes and estimation details ([Ou, Hunter, and Chow 2019](#)). Methodological tutorials demonstrate hierarchical stochastic differential equation models for affect dynamics, highlighting how drift and diffusion parameters map to psychological regulation and noise ([Oravecz, Tuerlinckx, and Vandekerckhove 2011](#)), and how short and long term timescales may be considered in the one model ([Boker, Daniel, and Orzek 2024](#)).

Linking theory to model requires explicit mappings from theoretical constructs to parameters, along with clearly stated scope conditions and levels of analysis, some general guidelines of which can be found in [Wilson and Collins \(2019\)](#). A range of works explore how formal models can serve, inform, or be tested against theories, and advocates using formal models as a practical toolkit for theory construction ([Haslbeck et al. 2022; Robinaugh et al. 2021](#)). Recent proposals to integrate intra- and inter-individual phenomena make these mappings explicit, clarifying which parameters encode within-person dynamics versus between-person structure and how each bears on theoretical claims ([Borsboom and Haslbeck 2024](#)). For panel designs, careful attention to time scale and confounding is crucial: cross-lagged effects are often misinterpreted when temporal assumptions are mismatched or when time-invariant confounders are not handled appropriately; guidance is available for diagnosing these issues and providing more defensible causal interpretations ([Driver 2025; Rohrer and Murayama 2023](#)). Finally, when theories posit continuous change, formalizing them as continuous-time systems can yield parameters (e.g., drift, diffusion) that align more directly with hypothesized mechanisms and developmental relations ([Driver and Tomasik 2023](#)).

Conclusion

Modern computational tools and software packages such as `ctsem` simplify the process of building intricate statistical models to meet the surge of interest in intensive longitudinal

data. However, the interpretation of complicated models still remains challenging despite the increasing ease of their implementation. In this tutorial, we have provided psychological researchers a set of tools to aid in the fit and interpretation of complex model structures, with flexibility to approximate many different theoretical structures of human dynamic systems.

Author Contributions

Charles C. Driver was responsible for all aspects of this work, including conceptualization, methodology, software development, validation, formal analysis, writing, and visualization. Michael Aristodemou contributed to drafting and reviewing the manuscript and provided contributions on methodology, validation, formal analysis.

Conflict of Interest

The authors declare that they have no conflicts of interest to disclose.

- Aalen, Odd Olai, Kjetil Røysland, Jon Michael Gran, Roger Kouyos, and Tanja Lange. 2016. “Can We Believe the DAGs? A Comment on the Relationship Between Causal DAGs and Mechanisms.” *Statistical Methods in Medical Research* 25 (5): 2294–314. <https://doi.org/10.1177/0962280213520436>.
- Asparouhov, Tihomir, Ellen L. Hamaker, and Bengt Muthén. 2018. “Dynamic Structural Equation Models.” *Structural Equation Modeling: A Multidisciplinary Journal* 25 (3): 359–88. <https://doi.org/10.1080/10705511.2017.1406803>.
- Bates, Douglas, Martin Mächler, Ben Bolker, and Steve Walker. 2015. “Fitting Linear Mixed-Effects Models Using lme4.” *Journal of Statistical Software* 67 (1): 1–48. <https://doi.org/10.18637/jss.v067.i01>.
- Berry, Daniel, and Michael T. Willoughby. 2017. “On the Practical Interpretability of Cross-Lagged Panel Models: Rethinking a Developmental Workhorse.” *Child Development* 88 (4): 1186–206. <https://doi.org/10.1111/cdev.12660>.
- Boker, Steven M., Katharine E. Daniel, and Jannik Orzek. 2024. “Separating Long-Term Equilibrium Adaptation from Short-Term Self-Regulation Dynamics Using Latent Differential Equations.” *Multivariate Behavioral Research* 59 (6): 1177–87. <https://doi.org/10.1080/00273171.2023.2228302>.
- Borsboom, Denny, and Jonas Haslbeck. 2024. “Integrating Intra- and Interindividual Phenomena in Psychological Theories.” *Multivariate Behavioral Research* 59 (6): 1290–1309. <https://doi.org/10.1080/00273171.2024.2336178>.
- Borsboom, Denny, Han L. J. van der Maas, Jonas Dalege, Rogier A. Kievit, and Brian D. Haig. 2021. “Theory Construction Methodology: A Practical Framework for Building Theories in Psychology.” *Perspectives on Psychological Science* 16 (4): 756–66. <https://doi.org/10.1177/1745691620969647>.

- Butler, Emily A., and Ashley K. Randall. 2013. “Emotional Coregulation in Close Relationships.” *Emotion Review* 5 (2): 202–10. <https://doi.org/10.1177/1754073912451630>.
- Carpenter, Bob, Andrew Gelman, Matthew D. Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. 2017. “Stan: A Probabilistic Programming Language.” *Journal of Statistical Software* 76 (1). <https://doi.org/10.18637/jss.v076.i01>.
- Curran, Patrick J., Khawla Obeidat, and Diane Losardo. 2010. “Twelve Frequently Asked Questions about Growth Curve Modeling.” *Journal of Cognition and Development* 11 (2): 121–36. <https://doi.org/10.1080/15248371003699969>.
- Driver, Charles C. 2023. “Inference with Cross-Lagged Effects – Common Causes in Dynamic Systems.” *PsyArXiv*, November. <https://doi.org/10.31234/osf.io/y3e9d>.
- . 2025. “Inference with Cross-Lagged Effects—Problems in Time.” *Psychological Methods* 30 (1): 174–202. <https://doi.org/10.1037/met0000665>.
- Driver, Charles C., Johan H. L. Oud, and Manuel C. Voelkle. 2017. “Continuous Time Structural Equation Modeling with R Package Ctsem.” *Journal of Statistical Software* 77 (5): 1–35. <https://doi.org/10.18637/jss.v077.i05>.
- Driver, Charles C., and Martin J. Tomasik. 2023. “Formalizing Developmental Phenomena as Continuous-Time Systems: Relations Between Mathematics and Language Development.” *Child Development* 94 (6): 1454–71. <https://doi.org/10.1111/cdev.13990>.
- Driver, Charles C., and Manuel C. Voelkle. 2018a. “Hierarchical Bayesian Continuous Time Dynamic Modeling.” *Psychological Methods* 23 (4): 774–99. <https://doi.org/10.1037/met0000168>.
- . 2018b. “Understanding the Time Course of Interventions with Continuous Time Dynamic Models.” In *Continuous Time Modeling in the Behavioral and Related Sciences*, edited by Kees van Montfort, Johan H. L. Oud, and Manuel C. Voelkle, 79–109. New York: Springer International Publishing. <http://www.springer.com/de/book/9783319772189>.
- Driver, Charles C., Manuel C. Voelkle, and Johan H. L. Oud. 2025. *Ctsem: Continuous Time Structural Equation Modelling*. Manual. CRAN. <https://cran.r-project.org/package=ctsem>.
- Durbin, James, and Siem Jan Koopman. 2012. *Time Series Analysis by State Space Methods*. 2nd ed. Oxford: Oxford University Press.
- Epskamp, Sacha. 2020. “Psychometric Network Models from Time-Series and Panel Data.” *Psychometrika* 85 (1): 206–31. <https://doi.org/10.1007/s11336-020-09697-3>.
- Gabry, Jonah, Daniel Simpson, Aki Vehtari, Michael Betancourt, and Andrew Gelman. 2019. “Visualization in Bayesian Workflow.” *Journal of the Royal Statistical Society: Series A* 182 (2): 389–402. <https://doi.org/10.1111/rssa.12378>.
- Geert, Paul van. 2011. “The Contribution of Complex Dynamic Systems to Development.” *Child Development Perspectives* 5 (4): 273–78. <https://doi.org/10.1111/j.1750-8606.2011.00197.x>.
- Gelman, Andrew, Xiao-Li Meng, and Hal Stern. 1996. “Posterior Predictive Assessment of Model Fitness via Realized Discrepancies.” *Statistica Sinica* 6 (4): 733–60. <https://www.jstor.org/stable/24306036>.
- Hamaker, Ellen L., Rebecca M. Kuiper, and Raoul P. P. P. Grasman. 2015. “A Critique of

- the Cross-Lagged Panel Model.” *Psychological Methods* 20 (1): 102–16. <https://doi.org/10.1037/a0038889>.
- Harvey, Andrew C. 1989. *Forecasting, Structural Time Series Models and the Kalman Filter*. Cambridge: Cambridge University Press.
- Haslbeck, Jonas M. B., Oisín Ryan, Donald J. Robinaugh, Lourens J. Waldorp, and Denny Borsboom. 2022. “Modeling Psychopathology: From Data Models to Formal Theories.” *Psychological Methods* 27 (6): 930–57. <https://doi.org/10.1037/met0000303>.
- Higham, Desmond J. 2001. “An Algorithmic Introduction to Numerical Simulation of Stochastic Differential Equations.” *SIAM Review* 43 (3): 525–46. <https://doi.org/10.1137/S0036144500378302>.
- Kalman, Rudolph E. 1960. “A New Approach to Linear Filtering and Prediction Problems.” *Journal of Basic Engineering* 82 (1): 35–45. <https://doi.org/10.1115/1.3662552>.
- Kenny, David A. 2005. “Cross-Lagged Panel Design.” In *Encyclopedia of Statistics in Behavioral Science*. John Wiley & Sons, Ltd. <https://doi.org/10.1002/0470013192.bsa156>.
- Kenny, David A., Deborah A. Kashy, and William L. Cook. 2006. *Dyadic Data Analysis*. New York: Guilford Press.
- Kloeden, Peter E., and Eckhard Platen. 1992. *Numerical Solution of Stochastic Differential Equations*. Berlin: Springer.
- Kuiper, Rebecca M, and Oisín Ryan. 2018. “Drawing Conclusions from Cross-Lagged Relationships: Re-considering the Role of the Time-Interval.” *Structural Equation Modeling: A Multidisciplinary Journal* 25 (5): 809–23. <https://doi.org/10.1080/10705511.2018.1431046>.
- Lütkepohl, Helmut. 2005. *New Introduction to Multiple Time Series Analysis*. Berlin: Springer.
- Nakagawa, Shinichi, and Holger Schielzeth. 2013. “A General and Simple Method for Obtaining R² from Generalized Linear Mixed-Effects Models.” *Methods in Ecology and Evolution* 4 (2): 133–42. <https://doi.org/10.1111/j.2041-210X.2012.00261.x>.
- Oravecz, Zita, Francis Tuerlinckx, and Joachim Vandekerckhove. 2011. “A Hierarchical Latent Stochastic Differential Equation Model for Affective Dynamics.” *Psychological Methods* 16 (4): 468–90. <https://doi.org/10.1037/a0024375>.
- Ou, Lu, Michael D. Hunter, and Sy-Miin Chow. 2019. “What’s for Dynr: A Package for Linear and Nonlinear Dynamic Modeling in R.” *The R Journal* 11 (1): 91–111. <https://doi.org/10.32614/rj-2019-012>.
- Robinaugh, Donald J., Jonas M. B. Haslbeck, Oisín Ryan, Eiko I. Fried, and Lourens J. Waldorp. 2021. “Invisible Hands and Fine Calipers: A Call to Use Formal Theory as a Toolkit for Theory Construction.” *Perspectives on Psychological Science* 16 (4): 725–43. <https://doi.org/10.1177/1745691620974697>.
- Rohrer, Julia M., and Kou Murayama. 2023. “These Are Not the Effects You Are Looking for: Causality and the Within-/Between-Persons Distinction in Longitudinal Data Analysis.” *Advances in Methods and Practices in Psychological Science* 6 (1): 25152459221140842. <https://doi.org/10.1177/25152459221140842>.
- Ryan, Oisín, and Ellen L. Hamaker. 2022. “Time to Intervene: A Continuous-Time Approach to Network Analysis and Centrality.” *Psychometrika* 87 (1): 214–52. <https://doi.org/10.1007/s11336-021-09767-0>.

- Voelkle, Manuel C., Johan H. L. Oud, Eldad Davidov, and Peter Schmidt. 2012. "An SEM Approach to Continuous Time Modeling of Panel Data: Relating Authoritarianism and Anomia." *Psychological Methods* 17 (2): 176–92. <https://doi.org/10.1037/a0027543>.
- Wickham, Hadley. 2011. "Ggplot2." *Wiley Interdisciplinary Reviews: Computational Statistics* 3 (2): 180–85.
- Wilson, Robert C, and Anne GE Collins. 2019. "Ten Simple Rules for the Computational Modeling of Behavioral Data." Edited by Timothy E Behrens. *eLife* 8 (November): e49547. <https://doi.org/10.7554/eLife.49547>.
- Yarkoni, Tal, and Jacob Westfall. 2017. "Choosing Prediction over Explanation in Psychology: Lessons from Machine Learning." *Perspectives on Psychological Science* 12 (6): 1100–1122. <https://doi.org/10.1177/1745691617693393>.