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

Charles C. Driver and Michael Aristodemou
Department of Psychology, University of Zurich

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.

Impact Statement

This tutorial shows how psychological researchers can use R and the ctsem package to build and understand dynamic systems models that capture the complexity of psychological processes. By providing a structured approach to model building, we aim to facilitate the translation of theoretical concepts into empirical models, ultimately enhancing our understanding of psychological concepts and their interrelations over time.

Keywords: dynamic systems, continuous time, hierarchical modelling, longitudinal data analysis, affect dynamics, ctsem, lme4, state-space models

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 & 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 et al., 2015), and related forms such as network vector autogression (Epskamp, 2020)). From there, one can

Charles C. Driver
Michael Aristodemou
The authors declare that they have no conflicts of interest to disclose. Author roles were classified using the Contributor Role Taxonomy (CRediT; https://credit.niso.org/) as follows: Charles C. Driver: conceptualization, methodology, software, validation, writing, visualization; Michael Aristodemou: formal analysis, writing, visualization
Correspondence concerning this article should be addressed to Charles C. Driver, Department of Psychology, University of Zurich, Binzmühlestrasse 14, Box 28, Zurich, Zurich 8057, Switzerland, Email: charles.driver@psychologie.uzh.ch

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; Driver & Tomasik, 2023; Ryan & Hamaker, 2022). 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 & 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.

Modelling Univariate Growth

Linear Change

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.

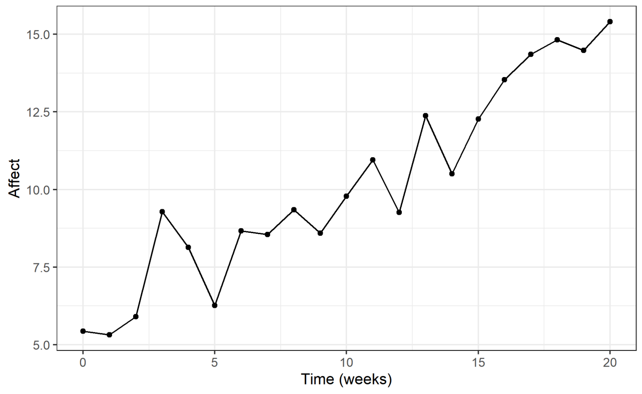
```r
# Define reusable plotting functions
plot_trajectory <- function(data, x="Time", y="Affect",
  x_label = "Time (weeks)", y_label = "Affect",
  color_var = NULL, show_legend = FALSE,  y2 = NULL){

  p <- ggplot(data, aes_string(x = x))

  # Handle single variable or couple plots
  if (is.null(y2)) {
    # Single variable plot
    p <- p + aes_string(y = y)
    p <- p + geom_point() + geom_line()
  } else {
    # Couple plot with two variables
    p <- p +
      geom_line(aes_string(y = y)) +
      geom_line(aes_string(y = y2), linetype = "dashed") +
      geom_point(aes_string(y = y))
  }
  if (!is.null(color_var)) {
    p <- p + aes_string(color = color_var)
  }

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

  if (!is.null(color_var) && is.null(y2)) {
    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 (`t0Affect <- 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 (`t0Affect`) 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.

```r
Time <- 0:20 # Generate time points for each measurement occasion
t0Affect <- 5 # Initial level of mood at week 0

# Specify linear model where affect increases by 0.51 each week
Affect <- t0Affect + 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_trajectory(data)
```

Figure 1

Generated Linear Affect Trajectory for a Single Person.



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 (`t0Affect <- 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 (`timeEffect <- rnorm(n = NSubjects, mean = 0.5, sd = 0.1) + scale(t0Affect) * -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.

```r
# Generate data for multiple subjects with individual differences
NSubjects <- 20
times <- seq(from=0, to=20, by=1) #times when subjects are measured
Nobs <- length(times) #number of observations per subject
#sample initial affect for each subject from normal distribution
t0Affect <- rnorm(n = NSubjects, mean = 5, sd = 2)
#sample rate of change in affect for each subject
timeEffect <- rnorm(n = NSubjects, mean = 0.5, sd = 0.1) +
  scale(t0Affect) * -0.2 #negative influence of initial affect

cor(t0Affect, timeEffect) #print correlation
```

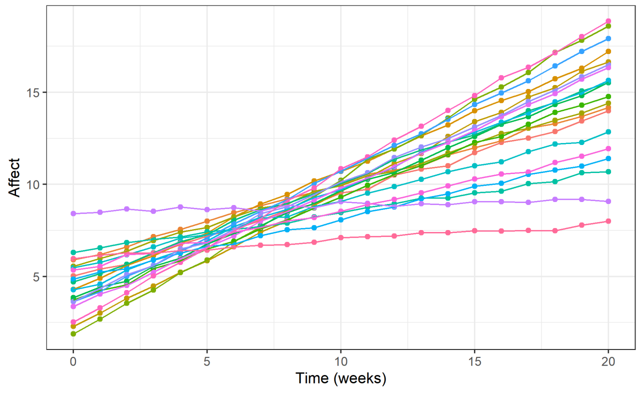
```
          [,1]
[1,] -0.9352227
```

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)`).

```r
#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 current row
for(subi in 1:NSubjects){
  for(obsi in 1:Nobs){ #for each observation of a subject
    row <- row + 1 #add 1 to row to reflect current row
    # current affect equals initial affect and effect of time:
    data$Affect[row] <- t0Affect[subi] +
      times[obsi] * timeEffect[subi]
    data$Time[row] <- times[obsi] #store time point for current row
    data$Subject[row] <- subi #store subject id for current row
  }
}
data$Affect <- data$Affect +
  rnorm(n=nrow(data), mean = 0, sd = .1) #measurement error

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

Figure 2

Affect Trajectories for 20 Subjects with Individual Differences in Initial Level and Rate of Change.



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):

```r
library(lme4)
lme_model <- lmer(Affect ~ # predict Affect
    1 + # a fixed effect for the intercept
    Time + # with a fixed effect of time
    #random intercept and random slope of time per subject:
    (1 + Time | 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.

```r
# 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: -501.7

Scaled residuals:
     Min       1Q   Median       3Q      Max
-2.43628 -0.68900  0.03258  0.62507  2.84218

Random effects:
 Groups   Name        Variance Std.Dev. Corr
 Subject  (Intercept) 2.469273 1.5714
          Time        0.050043 0.2237   -0.94
 Residual             0.008575 0.0926
Number of obs: 420, groups:  Subject, 20

Fixed effects:
            Estimate Std. Error t value
(Intercept)  4.53375    0.35148  12.899
Time         0.49787    0.05003   9.952

Correlation of Fixed Effects:
     (Intr)
Time -0.936
```

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 mislead 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.

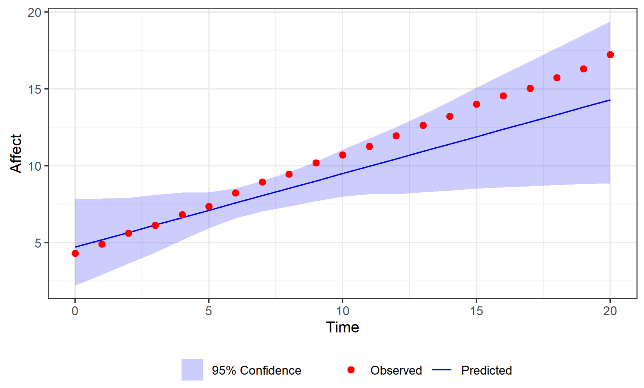
Visualizing Predictions

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.

```r
# 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 predictions, calculate mean and confidence intervals:
# Create new data frame including original data:
newdata <- data.frame(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")
```

Figure 3

LME4 Model Predictions and Confidence Interval Based on Parameter Estimates Only, Subject 3.

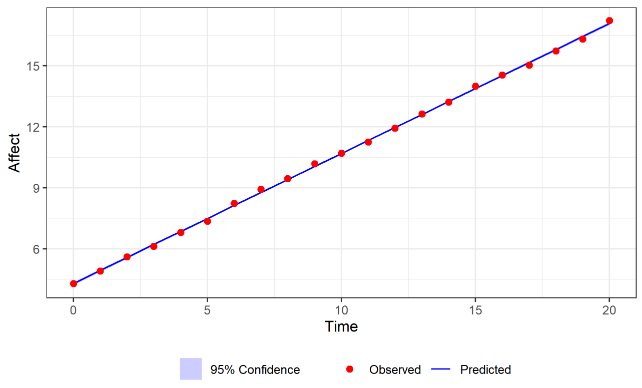


The first plot shows how well information solely from the estimated model parameters can reproduce our pattern of observations. The red dots show the observed affect values at each time point. The solid blue line shows the model implied expectation of affect, with the shaded blue area surrounding the line showing the variation / uncertainty about this trajectory. We can see that by solely relying on the model expectation, our model does a relativelypoor 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")
```

Figure 4

LME4 Model Predictions and 95% Confidence Interval for Subject 3 Conditional on All Observed Data.



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

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 T0, T1, and T2. 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 T0 by setting an initial level of affect to 5 ($\eta(0) = 5$). At T1, 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, T2, we can see that $0.8*6+2$ leads to current level of affect equal to 6.8. So from T0 to T1 our affect level grew by 1 unit, but from T1 to T2 it only grew by 0.8. Thus, the rate of growth is dampening as consequence of a change in the prior level of affect.

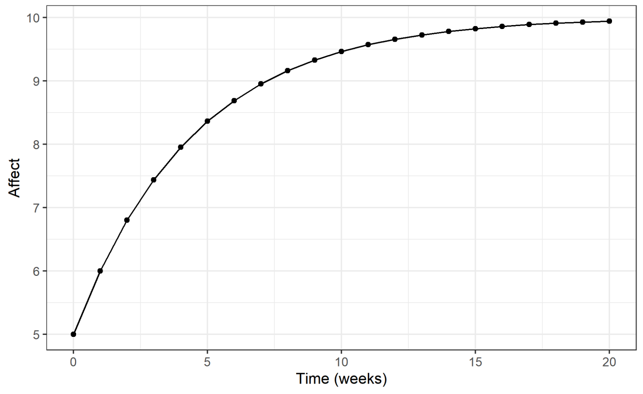
Simulation

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 (`t0Affect <- 5`). An if-else statement is used so that the first observation sets the initial level of affect (`if(i==1) Affect[i] <- t0Affect`) and each subsequent observation has an affect value that is generated by our model equation (`else Affect[i] <- A*Affect[i-1] + B`).

```r
times <- seq(from=0, to=20, by=1) #time points of measurement
A <- .8 #autoregressive coefficient / state dependence
t0Affect <- 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 first time point, set to initial affect:
  if(i==1) Affect[i] <- t0Affect
  #otherwise, use autoregression and continuous intercept:
  else Affect[i] <- A*Affect[i-1] + B
}

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

Figure 5

Nonlinear Trajectory from Discrete-Time Autoregressive Model.



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.

Modelling Univariate Dynamic Systems

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; Kuiper & Ryan, 2018; Voelkle et al., 2012).

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.

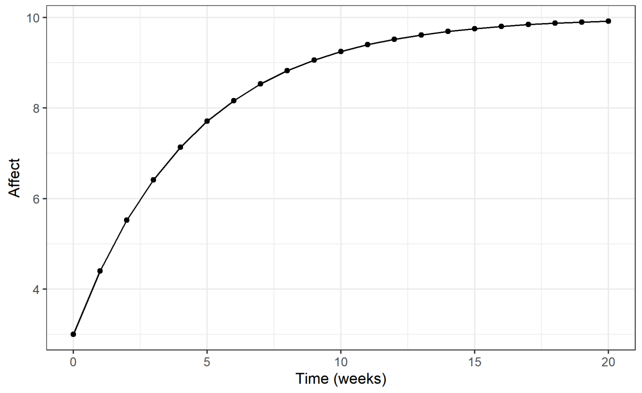
To generate data, we first need to define values for our model parameters. This includes: (1) the degree to which the current level of affect influences its own rate of change (`A`, continuous time state-dependence). (2) A constant input to the growth process (`B`, continuous intercept), and an initial level of affect (`t0Affect`). 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] <- t0Affect`). 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.

```r
times <- seq(from=0, to=20, by=1) #measurement times
Nobs <- length(times) # number of observations per subject
A <- -.2 # continuous time state dependence
```

```
t0Affect <- 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 first time point, set to initial affect
  if(i==1) Affect[i] <- t0Affect
  else{ # otherwise compute slope of affect at earlier time point
    dAffect <- A*Affect[i-1] + B
    # then update affect using slope and time step
    Affect[i] <- Affect[i-1] + dAffect * 1
  }
}

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

Figure 6

Generated Affect Trajectory from Continuous-Time Model.



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*Affect + 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) #time points of measurement
Nobs <- length(times) #number of observations per subject
A <- -.2 #continuous time state dependence
B <- .5 #continuous intercept
t0Affect <- 3
B <- 2 #continuous intercept
Affect <- rep(NA,Nobs) #create empty affect vector
Nsteps <- 10 #number of steps between each observation

for(i in 1:Nobs){ #for each time point
  #if first time point, set to initial affect
  if(i==1) Affect[i] <- t0Affect
  else{ #compute new affect state by sequence of small steps
    #initialise with state at previous time point
    Affect <- Affect[i-1]
    for(stepi in 1:Nsteps){ #take Nsteps for each observation
      #compute slope of affect at earlier time point
      dAffect <- A*Affect + B
      #update state using slope and time step
      Affect <- Affect + dAffect * 1/Nsteps
    }
    Affect[i] <- Affect
  }
}
```

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 processby 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 & 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

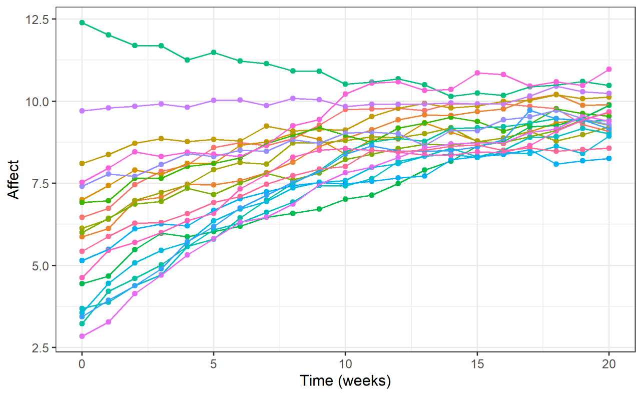
Adding system noise. To add system noise to our continuous time model, we will update the data generating code using the Euler method discussed in the previous section. The system noise term will be added to the affect level that is computed for each of our chosen time steps (`istep`). Thus, when we 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 (`t0Affect <- 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)`).

```
NSubjects <- 20
times <- seq(from=0, to=20, by=1) #time points of measurement
Nobs <- length(times) #number of observations per subject
t0Affect <- 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 #steps between each observation
row <- 0 #init current row tracker
for(subi in 1:NSubjects){
  for(obsi in 1:Nobs){ #for each observation of a subject
    row <- row + 1
    #if first time point, set to initial affect
    if(obsi==1) Affect <- t0Affect[subi]
    if(obsi>1){ #else compute new state via small steps in time
      for(stepi in 1:Nsteps){ #every step, do...
        dAffect <- A*Affect + B #compute affect slope
        Affect <- Affect + #new state = old state +
        dAffect * 1/Nsteps + #deterministic change +
          G * rnorm(n=1, mean=0, sd=sqrt(1/Nsteps)) #random change
      }
    }
    data$Affect[row] <- Affect #input affect data
    data$Time[row] <- times[obsi] #input time data
    data$Subject[row] <- subi #input subject data
  }
}
#add measurement error
data$Affect <- data$Affect+rnorm(n=nrow(data), mean = 0, sd = .05)
#plot trajectories
plot_trajectory(data, color_var = "as.factor(Subject)")
```

Figure 7

Affect Trajectories with System Noise and Individual Differences.



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 & Koopman, 2012; Harvey, 1989;

Kalman, 1960), allowing a measurement model to be linked to a latent dynamic model. The R package `ctsem` (Driver et al., 2017; Driver & 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("ctsem", dependencies=TRUE)
```

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 et al., 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 `t0Affect`. By adding `TRUE` behind `t0Affect||` 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 $Affect(t)$ 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), this is 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`.

```
m_basic <- ctModel( #define the ctsem model
  # Specify features of the data / model type
  manifestNames = "Affect", #names of observed variables in data
  latentNames = "Affect", #names for latent processes
  time = 'Time', #name of time column in dataset
  id = 'Subject', #name of subject column in dataset
  type='ct', #use continuous-time (dt for discrete-time)
  # Specify model components
  MANIFESTVAR = 'residualSD', #SD of measurement error
  LAMBDA = matrix(1,nrow=1,ncol=1), #factor loading matrix
  MANIFESTMEANS=0, #set measurement intercept / offset to 0
  CINT='B||FALSE', #continuous intercept with *no* random effects
  T0MEANS='t0Affect||TRUE', #initial affect + random effects
  DRIFT = 'stateDependence', #state dependence
  DIFFUSION = 'systemNoise') #system noise
```

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 represention of the equations we just specified by running the `ctModelLatex()` function, see Figure 8. 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.

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

```
f_basic <- ctStanFit(datalong = data, ctstanmodel = m_basic)
```

```
summary(f_basic, parmatrices= FALSE) #some output disabled
```

```
$residCovStd
       Affect
Affect  0.084

$resiCovStdNote
[1] "Standardised covariance of residuals"

$popsd
           mean     sd    2.5%     50% 97.5%
t0Affect 2.2979 0.3641 1.6466 2.2778  3.04

$popmeans
                   mean     sd    2.5%     50%   97.5%
t0Affect         4.5959 0.5075  3.6147  4.5982  5.5346
stateDependence -0.1016 0.0056 -0.1125 -0.1015 -0.0905
systemNoise      0.1963 0.0125  0.1722  0.1962  0.2205
residualSD       0.0543 0.0248  0.0206  0.0495  0.1134
B                1.0098 0.0436  0.9217  1.0123  1.0904

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

$loglik
[1] 36.1093

$npars
[1] 6

$aic
[1] -60.21859

$logposterior
[1] 36.1093

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

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

- **`$residCovStd`**: Standardized residual covariance, indicating how well the model explains the data - **`PARMATRICES`**: Parameter matrices for the model, suppressed in shown output. -**`$rawpopcorr`**: Correlations between random effects in the model – only a single random effect in this case, so not shown.
- **`$popsd`**: Population standard deviations for random effects in the model
- **`$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).

Figure 8

Matrix Representation of the Basic Univariate ctsem Model.

Subject parameter distribution:

$$\underbrace{\left[\text{t0Affect}_i\right]}_{\boldsymbol{\phi}(i)} \sim \text{tform}\left\{\text{N}\left(\left[\text{raw\_t0Affect}\right], \left[\text{rawPCov\_1\_1}\right]\right)\right\}$$

Initial latent state:

$$\underbrace{\left[\text{Affect}\right](t_0)}_{\boldsymbol{\eta}(t_0)} \sim \text{N}\left(\underbrace{\underbrace{\left[\text{t0Affect}\right]}_{}}_{\text{T0MEANS}}, \underbrace{\underbrace{UcorSDtoCov\left\{\left[1e-06\right]\right\}}_{\mathbf{Q}^*{}_{t0}}}_{\text{T0VAR}}\right)$$

Deterministic change:

$$\underbrace{\text{d}\left[\text{Affect}\right](t)}_{\text{d}\boldsymbol{\eta}(t)} = \left(\underbrace{\underbrace{\left[\text{stateDependence}\right]}_{\mathbf{A}}}_{\text{DRIFT}} \underbrace{\left[\text{Affect}\right](t)}_{\boldsymbol{\eta}(t)} + \underbrace{\underbrace{\left[\text{B}\right]}_{\mathbf{b}}}_{\text{CINT}}\right)\text{d}t \quad +$$

Random change:

$$\underbrace{\underbrace{UcorSDtoChol\left\{\left[\text{systemNoise}\right]\right\}}_{\mathbf{G}}}_{\text{DIFFUSION}} \underbrace{\text{d}\left[W_1\right](t)}_{\text{d}\mathbf{W}(t)}$$

Observations:

$$\underbrace{\left[\text{Affect}\right](t)}_{\mathbf{Y}(t)} = \underbrace{\underbrace{\left[1\right]}_{\boldsymbol{\Lambda}}}_{\text{LAMBDA}} \underbrace{\left[\text{Affect}\right](t)}_{\boldsymbol{\eta}(t)} + \underbrace{\underbrace{\left[0\right]}_{\boldsymbol{\tau}}}_{\text{MANIFESTMEANS}} \quad +$$

Observation noise:

$$\underbrace{\underbrace{UcorSDtoChol\left\{\left[\text{residualSD}\right]\right\}}_{\boldsymbol{\Theta}}}_{\text{MANIFESTVAR}} \underbrace{\left[\epsilon_1\right](t)}_{\boldsymbol{\epsilon}(t)}$$

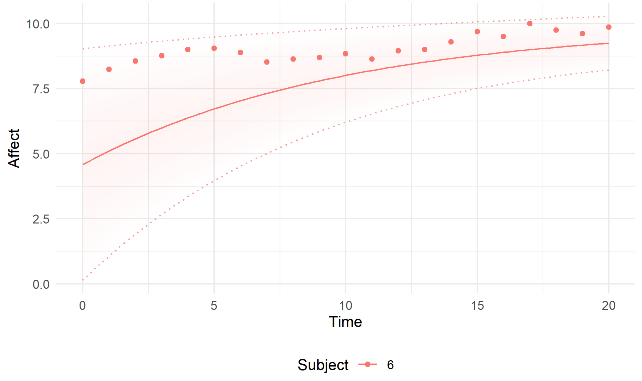
System noise distribution per time step: $\Delta\left[W_{j\in[1,1]}\right](t-u) \sim \text{N}(0, t-u)$ Observation noise distribution: $\left[\epsilon_{j\in[1,1]}\right](t) \sim \text{N}(0,1)$

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

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

Figure 9

ctsem Predictions for Subject 6 Based on Parameter Estimates Only.

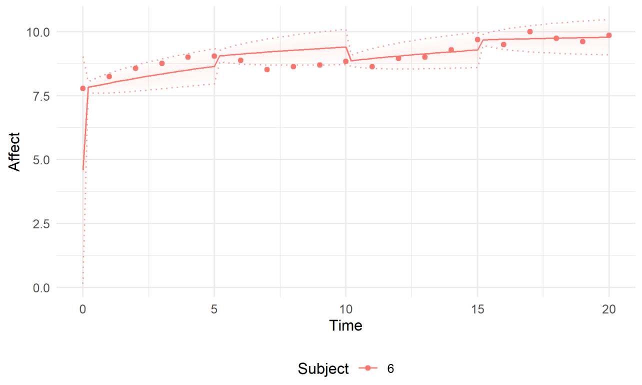


Just like in the case of our simple linear model, we see that the model does a poor job at predicting the observations (red dots) and there is a large degree of uncertainty (shaded red ribbon) around the model-implied trajectory (solid red line). Now let's allow the model to learn from every 5th observation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

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

Figure 10

ctsem Predictions for Subject 6 Conditional on Parameter Estimates and Every 5th Observation.

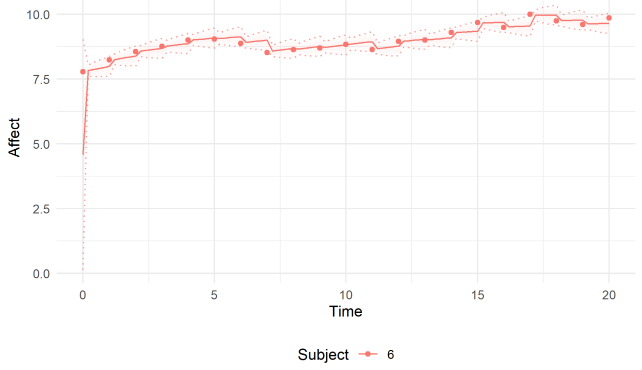


We can see that the model already does a better job at predicting the observations, and the uncertainty in its predicted trajectory is reduced (though growing larger again towards the end of the time period before a new observation is fed into the model). Lastly, let us include all past and present observations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

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

Figure 11

ctsem Predictions for Subject 6 Conditional on All Data.



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

Time-Dependent Predictors: Therapy Sessions as Impulse Effects (Continuous Time)

So far, we have modeled individual differences using time-independent predictors — variables that remain constant across time for each person (like personality traits or relationship duration). However, many influences on psychological processes vary within persons over time. For example, therapy sessions, stressful life events, medication doses, or social interactions can all have time-varying effects on affect. These are captured using *time-dependent predictors* (TDpreds).

In `ctsem`, time-dependent predictors operate (by default) as *impulse effects*: they provide an instantaneous "shock" to the latent process at the moment they occur. Critically, the effect of this impulse then feeds through the system over time, just as with any other inputs (such as the random fluctuations we've already introduced). In self-regulating systems, where inputs tend to dissipate back towards equilibrium, such an impulse effect will also gradually decay back toward the equilibrium. This provides a natural model for interventions that have temporary effects — like a therapy session that may temporarily lift affect, but then fades over time.

Consider Jill, who attends weekly therapy sessions during her 20-week treatment period. Each session provides a temporary boost to her affect. The longer term effect of the therapy is already accounted for by the growth trajectory we have already specified in earlier examples. Between sessions, the short term effect of therapy fades away and her affect gradually returns toward her general growth trajectory. The key insight is that we do not need to separately model the "decay" of the therapy effect — it emerges naturally from the state dependence dynamics we have already specified.

Mathematically, if a therapy session occurs at time t_k with effect size M, the latent state receives an instantaneous impulse:

$$\eta(t_k^+) = \eta(t_k^-) + M$$

where $\eta(t_k^-)$ is the state just before the session and $\eta(t_k^+)$ is the state just after.

Simulation

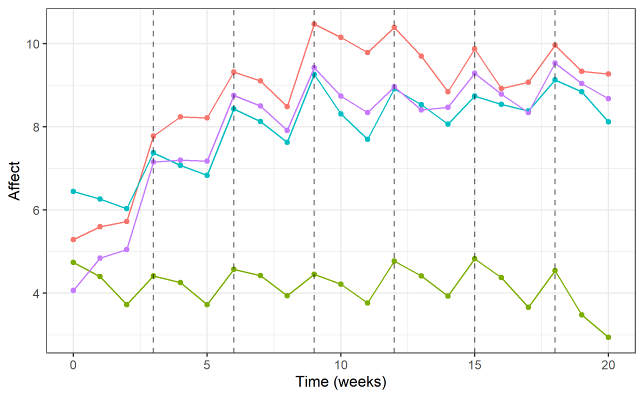
To simulate data with therapy sessions as impulse effects, we create a time-dependent predictor `Therapy` that equals 1 at observations when a therapy session occurs and 0 otherwise. For simplicity, we schedule therapy sessions at regular intervals (every 3 time units).

```
# Generate data for multiple subjects with therapy sessions
NSubjects <- 20
times <- seq(from=0, to=20, by=1) #time points of measurement
Nobs <- length(times) #number of observations per subject
t0Affect <- rnorm(n = NSubjects, mean = 5, sd = 2)
A <- -.2 #continuous time state dependence
B <- rnorm(n = NSubjects, mean = 1, sd = .3) #CINT
G <- .3 #system noise coefficient
M <- 1.5 #therapy impulse effect (positive = boost to affect)

# Define therapy sessions (every 3 time units, starting at time 3)
therapyTimes <- seq(from=3, to=max(times), by=3)

#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),
  Therapy = rep(NA, NSubjects*Nobs))

Nsteps <- 10 # steps between observations
```

```
row <- 0 #initialize row counter
for(subi in 1:NSubjects){
  for(obsi in 1:Nobs){
    row <- row + 1
    currentTime <- times[obsi]
    therapyToday <- ifelse(currentTime %in% therapyTimes, 1, 0)

    if(obsi == 1) Affect <- t0Affect[subi]
    if(obsi > 1){ #update state via small steps in time
      for(stepi in 1:Nsteps){
        dAffect <- A * Affect + B[subi] #deterministic slope
        #update state using slope and time step
        Affect <- Affect + dAffect * 1/Nsteps +
          G * rnorm(n=1, mean=0, sd=sqrt(1/Nsteps)) #system noise
      }
    }
    #add impulse if therapy session (dummy variable)
    Affect <- Affect + M * therapyToday
    data$Affect[row] <- Affect #add affect to data
    data$Time[row] <- currentTime #add time to data
    data$Subject[row] <- subi #add subject to data
    data$Therapy[row] <- therapyToday #add therapy to data
  }
}

#add measurement error (after creating process!)
data$Affect <- data$Affect+rnorm(n=nrow(data), mean = 0, sd = .1)

plot_trajectory(data[data$Subject %in% 1:4, ], #plot
  color_var = "as.factor(Subject)") + #then overlay therapy
  geom_vline(xintercept=therapyTimes,linetype="dashed", alpha=.5)
```

Figure 12

Affect Trajectories for Four Subjects with Therapy Sessions as Impulse Effects (Vertical Lines).



The dashed vertical lines indicate therapy sessions. Notice how affect tends to spike upward at these times and then follows the general growth trajectory we specified in earlier examples.

Model Fitting

To fit this model in `ctsem`, we add the `Therapy` variable as a time-dependent predictor using the `TDpredNames` argument. By default, `ctsem` estimates the effect of time-dependent predictors on each latent process, here we explicitly name the parameter `td_Affect_Therapy`. The estimated effect represents the instantaneous impulse — the immediate impulse to the latent state when the predictor equals 1. We will not estimate individual differences in this effect, but one could imagine such differences being of interest if going further with the model.

```
m_therapy <- ctModel(
  manifestNames = "Affect",
  latentNames = "Affect",
  TDpredNames = "Therapy", #time-dependent predictor
  TDPREDEFFECT = c('td_Affect_Therapy'),
  time = 'Time',
  id = 'Subject',
  type = 'ct',
  MANIFESTVAR = 'residualSD',
  LAMBDA = matrix(1, nrow=1, ncol=1),
  MANIFESTMEANS = 0,
  CINT = 'B||TRUE', #continuous intercept with random effects
  T0MEANS = 't0Affect||TRUE',
  DRIFT = 'stateDependence',
  DIFFUSION = 'systemNoise')
```

```
f_therapy <- ctStanFit(datalong = data, ctstanmodel = m_therapy)
```

```
s <- summary(f_therapy)
print(s$popmeans)
```

```
                     mean     sd    2.5%     50%   97.5%
t0Affect           6.1443 0.4074  5.3690  6.1533  6.9627
stateDependence   -0.2052 0.0181 -0.2415 -0.2045 -0.1706
systemNoise        0.2955 0.0218  0.2552  0.2950  0.3398
residualSD         0.0928 0.0342  0.0438  0.0871  0.1750
td_Affect_Therapy  1.4975 0.0351  1.4291  1.4972  1.5702
B                  1.0285 0.1559  0.7033  1.0317  1.3218
```

The `td_Affect_Therapy` parameter represents the estimated impulse effect of therapy sessions. A positive value indicates that therapy boosts affect. Our simulated value was `M = 1.5`, so we expect an estimate close to this.

Modelling Multivariate Systems

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 et al., 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. *Correlation in system noise*, 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 correlation in the noise — capturing the idea that Jill and Bert each experience unique influences, but also some shared influences.

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

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

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

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 ($A_{12}\eta_2(t)$ — the cross-effect from partner), (3) a constant input (B_1), and (4) random noise that may affect both partners ($G_{11}dW_1(t)+G_{12}dW_2(t)$). In general, we would not be able to estimate G_{12} distinctly from G_{21} and vice versa, as they are correlated, so when it comes to estimation we will estimate a single parameter to represent the correlation between them, as well as the standard deviations of the noise for each partner. 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 ($A_{21}\eta_1(t)$), 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 ($A_{12}$ and A_{21}) capture how partners influence each other's rate of change. If A_{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 correlated-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*Affect1 + Across * Affect2 + B`). (2) We allow the random fluctuations in the couple's affect process to covary. We do this by generating correlated noise according to a specified `DIFFUSIONcov` covariance matrix, where the diagonal elements specify the variance for each process and the off-diagonal elements specify the covariance between processes.

```r
NSubjects <- 20
times <- seq(from=0, to=40, by=1) #time points of measurement
Nobs <- length(times) #number of observations per subject
t0Affect1 <- rnorm(n = NSubjects, mean = 5, sd = 2)
t0Affect2 <- rnorm(n = NSubjects, mean = 5, sd = 2)
A <- -.2 #continuous time state dependence
Across <- .1 #cross-effect state dependence
B <- 1 #continuous intercept
#covariance matrix for system noise
DIFFUSIONcov <- matrix(c(1, .05, .05, .1), nrow=2, ncol=2)

#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 two individuals

Nsteps <- 100 #steps between observations

row <- 0 # initialize row counter
for(subi in 1:NSubjects){
  for(obsi in 1:Nobs){ #for each observation of a subject
    row <- row + 1 #increment row counter
    if(obsi==1){ #set initial affect
      Affect1 <- t0Affect1[subi]
      Affect2 <- t0Affect2[subi]
    }
    if(obsi>1){ #update state via small steps in time
      for(stepi in 1:Nsteps){ #for each step
        # compute deterministic slopes of affect
        dAffect1 <- A*Affect1 + Across * Affect2 + B
        dAffect2 <- A*Affect2 + Across * Affect1 + B

        # generate correlated noise
        systemNoise <- MASS::mvrnorm(n=1, mu=c(0,0),
          Sigma=DIFFUSIONcov/Nsteps) #scaled by stepsize

        #update state using slopes and time step + noise
        Affect1 <- Affect1 + dAffect1/Nsteps + systemNoise[1]
        Affect2 <- Affect2 + dAffect2/Nsteps + systemNoise[2]
      }
    }
    data$Affect1[row] <- Affect1 #input affect data
    data$Affect2[row] <- Affect2 #input affect data
    data$Time[row] <- times[obsi] #input time data
    data$Subject[row] <- subi #input subject data
  }
}
#add measurement error
data$Affect1 <- data$Affect1+rnorm(n=nrow(data),mean = 0,sd=.05)
data$Affect2 <- data$Affect2+rnorm(n=nrow(data),mean = 0,sd=.05)
```

```
data$Couple <- factor(data$Subject) #factor subject for plotting
plot_trajectory(data[data$Subject %in% c(1,2),],y="Affect1",
  show_legend=TRUE, y2="Affect2",color_var="Couple")
```

Figure 13

Affect Trajectories for Both Partners in One Couple.

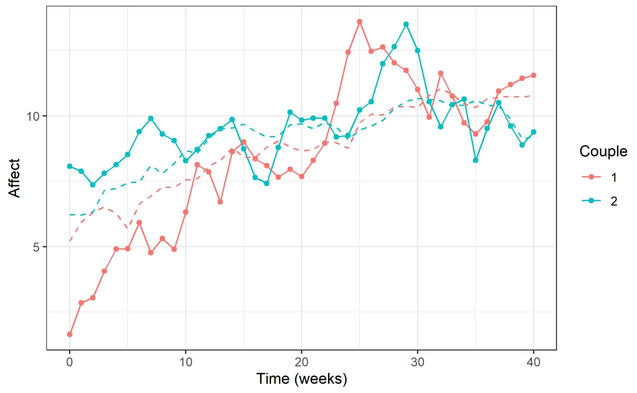


Figure 13 shows the affect trajectories for two couples. Each partner in the couple is clearly distinct, but couples show similarities in comparison to other couples, due to both shared influences and effects from one partner on the other.

Model Fitting

First, the model syntax for ctsem for the couple model:

```
m_couple <- ctModel( #define the ctsem model
  manifestNames = c("Affect1",'Affect2'), #observed variables
  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 (dt for discrete-time)
  MANIFESTVAR = c( #covariance matrix of measurement error
    'residualSD1',0, #sd on diagonal, 0 correlation assumed.
    0, 'residualSD2'),
  LAMBDA = diag(1,2), #diagonal factor loading matrix (identity)
  MANIFESTMEANS=c(0,0), #zero measurement intercept / offset
  CINT=c('B1||F','B2||F'), #continuous intercept no random effects
  TOMEANS=c('t0Affect1||TRUE',
  't0Affect2||TRUE'), #initial affect with random effects
  DRIFT = c(
    'auto1','cross12', #auto effect partner 1, cross-effect 2 to 1
    'cross21','auto2'), #cross-effect 1 to 2, auto effect for 2
  DIFFUSION = c(
    'systemNoise1', 0, #system noise for 1, 0 in upper triangle
    'systemNoiseCor', 'systemNoise2')) #corr, and sd for 2
```

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 expanded matrix form to more easily link it with the more compact syntax of ctsem. We use ctModelLatex(m_couple) for this purpose, shown in Figure 14. Here, both partners' equations are combined into matrix equations.

The first matrix equations in Figure 14 show the subject parameter distribution, confirming that we have two parameters in our model that vary across individuals, t0Affect1 and t0Affect2, and these are assumed normally distributed according to some estimated mean and covariance, *possibly* with some transformation applied after this distribution to ensure parameters remain within necessary bounds (primarily relevant when e.g., correlation or standard deviation parameters are estimated, see the ctsem manual for more details on this).

The second matrix equation in Figure 14 refers to the distribution of the initial latent state **within-subject** – because we have individual differences in the initial states, these are not necessary (not uniquely estimable), so fixed to a small variance and determined based on the subject specific parameters t0Affect1 and t0Affect2. The TOMEANS argument is used to specify these. By adding TRUE behind t0Affect1|| and t0Affect2|| we are allowing for random effects (between-subject variance) around the population estimate of the initial level of affect in our sample.

Next we look at the dynamic model component of Figure 14, made up of the deterministic change and random change lines. On the left, the vector $(d\text{Affect}_1, d\text{Affect}_2)$ represents 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 $(A_{12},\ A_{21})$ are cross-effects (how one partner affects the other's change). The DRIFT argument to ctModel specifies this matrix. The CINT (continuous intercept) vector contains the constants (B_1, B_2) that are steady inputs for each partner, and is specified by the CINT argument to ctModel. Because in this case we only estimate a fixed effect (one value for all subjects), we switch off random effects (on by default for initial values and intercept parameters, i.e., TOMEANS, CINT, MANIFESTMEANS) by adding FALSE behind B||, i.e. B||FALSE.

Turning to the random change in Figure 14, it is created by a complex function (UcorSDtoChol) that takes the $DIFFUSION$ matrix as input, where the diagonal parameters are standard deviations and the lower-triangle off-diagonal parameters represent 'unconstrained correlations' between the noise terms. The ctModel function takes DIFFUSION as an argument to specify this matrix of standard deviations and unconstrained correlations. Our standard deviation parameters are systemNoise1 and systemNoise2 and the correlation parameter is called systemNoiseCor. These unconstrained correlation parameters are unfortunately somewhat complex to interpret directly, but are necessary to allow for continuous-moderation and random-effects. For interpretation, obtaining the DIFFUSION covariance from the summary() or ctStanContinuousPars() output is recommended, possibly with a conversion to a correlation matrix via the cov2cor() function.

The latent process model is linked to our observations us-

Figure 14

Matrix Equations for the Bivariate Couple ctsem Model.

Subject parameter distribution:
$$\underbrace{\begin{bmatrix}\text{t0Affect1}_i\\ \text{t0Affect2}_i\end{bmatrix}}_{\boldsymbol{\phi}(i)} \sim \text{tform}\left\{\text{N}\left(\begin{bmatrix}\text{raw\_t0Affect1}\\ \text{raw\_t0Affect2}\end{bmatrix}, \begin{bmatrix}\text{rawPCov\_1\_1} & \text{rawPCov\_2\_1}\\ \text{rawPCov\_2\_1} & \text{rawPCov\_2\_2}\end{bmatrix}\right)\right\}$$

Initial latent state:
$$\underbrace{\begin{bmatrix}\text{Affect1}\\ \text{Affect2}\end{bmatrix}(t_0)}_{\boldsymbol{\eta}(t_0)} \sim \text{N}\left(\underbrace{\begin{bmatrix}\text{t0Affect1}\\ \text{t0Affect2}\end{bmatrix}}_{\underbrace{}_{\text{T0MEANS}}}, \underbrace{UcorSDtoCov\left\{\begin{bmatrix}1e-06 & 0\\ 0 & 1e-06\end{bmatrix}\right\}}_{\underbrace{\mathbf{Q}^{*}{}_{t0}}_{\text{T0VAR}}}\right)$$

Deterministic change:
$$\underbrace{\text{d}\begin{bmatrix}\text{Affect1}\\ \text{Affect2}\end{bmatrix}(t)}_{\text{d}\boldsymbol{\eta}(t)} = \left(\underbrace{\begin{bmatrix}\text{auto1} & \text{cross12}\\ \text{cross21} & \text{auto2}\end{bmatrix}}_{\underbrace{\mathbf{A}}_{\text{DRIFT}}}\underbrace{\begin{bmatrix}\text{Affect1}\\ \text{Affect2}\end{bmatrix}(t)}_{\boldsymbol{\eta}(t)} + \underbrace{\begin{bmatrix}\text{B1}\\ \text{B2}\end{bmatrix}}_{\underbrace{\mathbf{b}}_{\text{CINT}}}\right)\text{d}t \quad +$$

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

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

Observation noise:
$$\underbrace{UcorSDtoChol\left\{\begin{bmatrix}\text{residualSD1} & 0\\ 0 & \text{residualSD2}\end{bmatrix}\right\}}_{\underbrace{\boldsymbol{\Theta}}_{\text{MANIFESTVAR}}}\underbrace{\begin{bmatrix}\epsilon_1\\ \epsilon_2\end{bmatrix}(t)}_{\boldsymbol{\epsilon}(t)}$$

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

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

ing a (linear) measurement model, given by the observation equation in Figure 14. This equation also comprises a deterministic component and a random component, and links what we actually observe ($\text{Affect}_1(t)$, $\text{Affect}_2(t)$) to the underlying latent processes ($\text{Affect}_1(t)$, $\text{Affect}_2(t)$) that represent 'true' affect.

The deterministic component of the measurement model is given by $\Lambda\eta(t)+\tau$, where Λ (`LAMBDA` argument to `ctModel`) 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 (`MANIFESTMEANS` argument to `ctModel`) 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 (This becomes particularly useful when estimating factor models, but can also be estimated instead of `CINT`, such that latent processes converge towards zero and `MANIFESTMEANS` offsets this an appropriate amount for the data).

The random component of the measurement model (observation noise) is given by $\epsilon(t)$, where ϵ is a multivariate normal distribution with mean 0 and (usually diagonal) covariance matrix Θ. The Θ matrix (`MANIFESTVAR` argument to `ctModel`) contains measurement error standard deviations (residualSD1, residualSD2) — 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 here set to 0 (but could be estimated if desired, as with the `DIFFUSION` correlations).

An important note when specifying the system matrices in `ctsem`, is that they can be explicitly created using the `matrix()` function in R, but can also be passed in as a vector (except for the LAMBDA, factor-loading matrix) using the `c()` function, and this vector will be read in **row-wise** in contrast to typical R convention (this makes model specification more intuitive to read as it aligns with the matrix representation).

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, with the exception of code showing the correlation in system noise.

```
f_couple <- ctStanFit(datalong = data, ctstanmodel = m_couple)
parmatrices <- ctStanContinuousPars(f_couple) #estimated matrices
print(cov2cor(parmatrices$DIFFUSIONcov)) #system noise correlation
```

```
           Affect1    Affect2
Affect1 1.00000000 0.07371448
Affect2 0.07371448 1.00000000
```

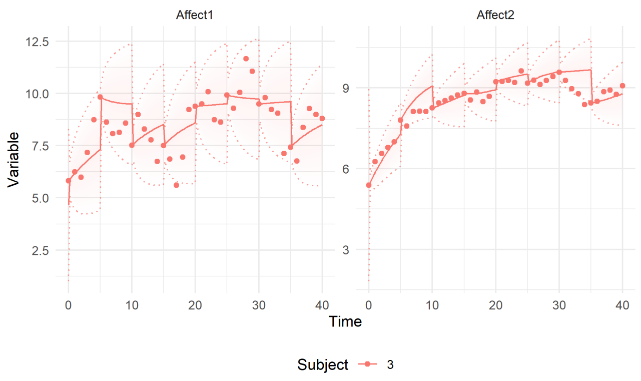
Visualize Predictions

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

Figure 15

ctsem Predictions for Both Partners in Dyad 3 Conditional on Every 5th Observation.



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 Jilly — 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(f_couple,plot=T)
```

Figure 16

Impulse Response Functions Assuming Independent Shocks

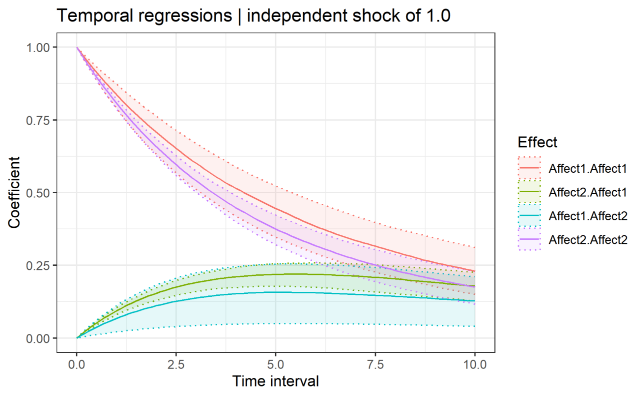
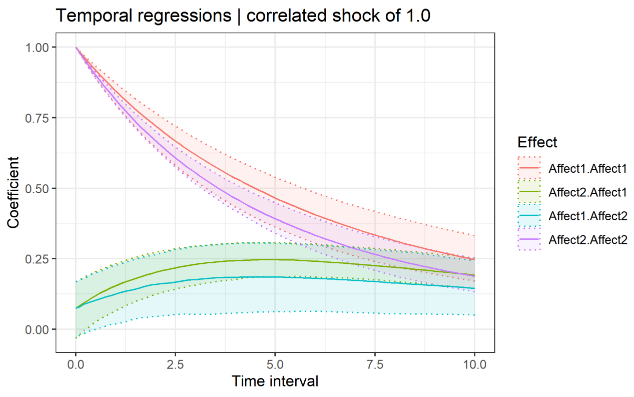


Figure 17

Impulse Response Function Assuming Correlated Shocks



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 paramters 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(f_couple,plot=T, observational=TRUE)
```

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 et al., 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 <- Bcommon + rnorm(n = NSubjects, mean = 2, sd = .3)`).

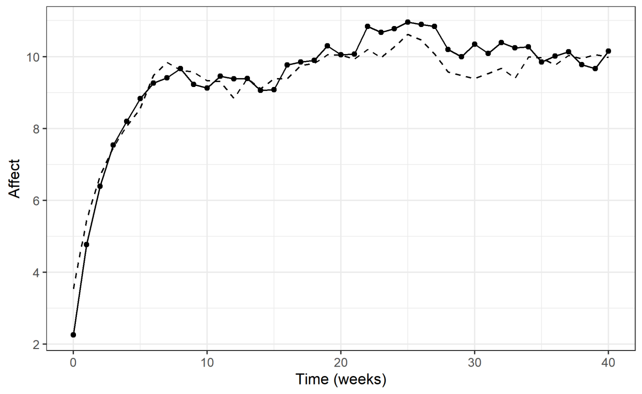
To add individual differences in the continuous intercept parameter, we index the parameter to denote that it is specific to the denoted subject. So `B1` becomes `B1[subi]` where `subi` refers to the subject identifier.

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 <- rnorm(n= NSubjects, mean = .1, sd=.05) + .05*TogetherZ`. The cross-effect parameter is also indexed in our data generating loop to generate a subject-specific parameter, e.g. `Across[subi] * Affect2`.

```r
NSubjects <- 20
times <- seq(from=0, to=40, by=1) #time points of measurement
Nobs <- length(times) #number of observations per subject
t0Affect1 <- rnorm(n = NSubjects, mean = 5, sd = 2)
t0Affect2 <- rnorm(n = NSubjects, mean = 5, sd = 2)
Together <- runif(n=NSubjects, min=0, max=20) #years together
TogetherZ <- scale(Together) #standardise time together
A <- -.4 #continuous time state dependence
#cross-effect state dependence -- affected by time together.
Across <- rnorm(n= NSubjects, mean = .1, sd=.05) + .05*TogetherZ

# generate continuous intercepts for each individual,
# assuming high affect individuals may partner with high affect
# common continuous intercept variance:
Bcommon <- rnorm(n = NSubjects, mean = 1, sd = .2)
B1 <- Bcommon + rnorm(n = NSubjects, mean = 2, sd = .3) #partner 1
B2 <- Bcommon + rnorm(n = NSubjects, mean = 2, sd = .3) #partner 2

# covariance matrix for system noise
DIFFUSIONcov <- matrix(c(.1, .05, .05, .1), nrow=2, ncol=2)

#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 affect for two individuals

Nsteps <- 100 # steps between each observation
row <- 0 #init current row tracker
for(subi in 1:NSubjects){
  for(obsi in 1:Nobs){ #for each observation of a subject
    row <- row + 1
    if(obsi==1){ # if first time point, set to initial affect
      Affect1 <- t0Affect1[subi]
      Affect2 <- t0Affect2[subi]
    }
    if(obsi>1){ # else update state via small steps in time
      for(stepi in 1:Nsteps){ # for each step

        #compute deterministic slopes
        dAffect1 <- A*Affect1 + Across[subi] * Affect2 + B1[subi]
        dAffect2 <- A*Affect2 + Across[subi] * Affect1 + B2[subi]

        # generate correlated noise scaled by stepsize
        systemNoise <- MASS::mvrnorm(n=1, mu=c(0,0),
          Sigma=DIFFUSIONcov/Nsteps)

        #update states using slope and time step, add noise
        Affect1<- Affect1 + dAffect1 * 1/Nsteps+systemNoise[1]
        Affect2<- Affect2 + dAffect2 * 1/Nsteps+systemNoise[2]
      }
    }
    data$Affect1[row] <- Affect1 #input affect data
    data$Affect2[row] <- Affect2 #input affect data
    data$Time[row] <- times[obsi] #input time data
    data$Subject[row] <- subi #input subject data
    data$TogetherZ[row] <- TogetherZ[subi] #input time together
  }
}

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

plot_trajectory(data[data$Subject==1,],y="Affect1",y2="Affect2")
```

Figure 18

Affect Trajectories for Both Partners in One Couple with Time-Together Moderation.



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 restricted to linear effects at present (a complication here is that parameters such as correlations or standard deviations have non-linear transforms applied afterwards, to ensure they cannot go out of bounds — see the ctsem manual for more detail). 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 & 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 & 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 (TogetherZ) as a stable source of individual differences (time-independent predictor). This is done using the argument TIpredNames = c('TogetherZ'). 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 TogetherZ 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||TogetherZ. This would estimate the cross21 parameter, allow it to vary as a linear function of TogetherZ, 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 TogetherZ. 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).

```r
# Fit continuous time structural equation model
m_indiff <- ctModel( #define the ctsem model
  tipredDefault = FALSE, #moderation disabled unless specified
  TIpredNames = c('TogetherZ'), #time independent predictors
  manifestNames = c("Affect1",'Affect2'), #observed variables
  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
  MANIFESTVAR = c('residualSD1',0,
    0, 'residualSD2'), #sd's of the residual / measurement error
  LAMBDA = diag(1,2), #factor loading matrix
  MANIFESTMEANS=c(0,0), #no measurement intercept / offset
  CINT=c('B1||TRUE','B2||TRUE'), #cint with random effects
  TOMEANS=c('t0Affect1||TRUE','t0Affect2||TRUE'), #initial affect
  DRIFT = c( #auto and cross effects matrix, cross effects with...
    'auto1', 'cross21||TRUE||TogetherZ', #random effects and...
    'cross21||TRUE||TogetherZ','auto2' ), #moderation
  DIFFUSION = c( #system noise matrix
    'systemNoise1', 0, #sd for subj 1, 0 in upper triangle
    'systemNoiseCor', 'systemNoise2')) #correlation, sd for subj 2
```

```r
f_indiff <- ctStanFit(datalong = data, ctstanmodel = m_indiff)
```

```r
s = summary(f_indiff) #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_TogetherZ_cross21 0.0456 0.0111 0.0246 0.0454 0.0669 4.0961
```

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
t0Affect2__t0Affect1  0.22 0.31 -0.30  0.14  0.91  0.69
cross21__t0Affect1    0.08 0.45 -0.81  0.21  0.74  0.19
B1__t0Affect1        -0.09 0.48 -0.73 -0.22  0.95 -0.18
B2__t0Affect1         0.01 0.45 -0.65 -0.11  0.94  0.03
cross21__t0Affect2   -0.05 0.35 -0.77 -0.01  0.54 -0.15
B1__t0Affect2        -0.15 0.45 -0.76 -0.28  0.92 -0.33
B2__t0Affect2        -0.05 0.44 -0.70 -0.18  0.92 -0.12
B1__cross21          -0.21 0.32 -0.79 -0.16  0.41 -0.64
B2__cross21          -0.07 0.40 -0.82  0.01  0.64 -0.18
B2__B1                0.23 0.38 -0.47  0.20  0.94  0.61
```

Visualize Individual Differences

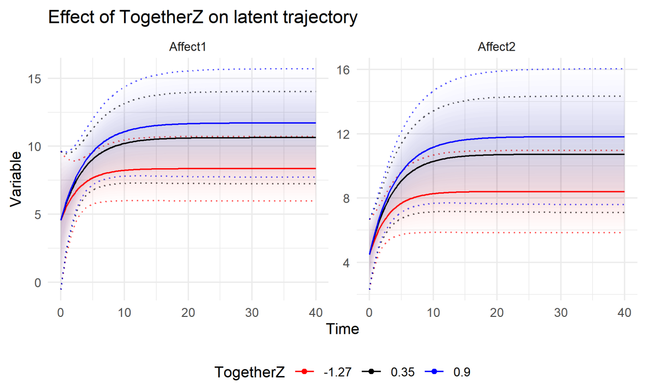
We can create plots of the trajectory of affect implied by the estimated model parameters, conditional on +/-1 standard deviation and median of any time-independent predictors included in the model using the ctPredictTIP function.

The first set of plots created, stored under the $Process subobject, includes both latent process and predicted observation trajectories of partner 1 (Affect1) and partner 2 (Affect2). 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. We plot the latent trajectories only here:

```
tipredplots <- ctPredictTIP(f_indiff, plot=T, tipreds = c('TogetherZ')) #create list of plots
print(tipredplots$Process$Latent) #latent process predictions
```

Figure 19

Predicted Latent Affect Trajectories by Time Spent Together (TogetherZ).

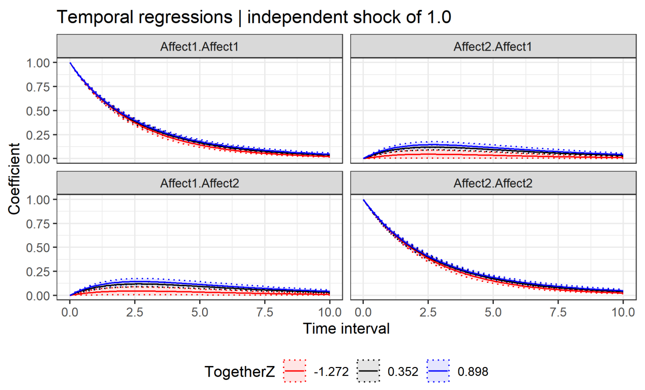


The model implied dynamics, and how these differ with regard to the covariate, are stored under the $Dynamics subobject. The first of these shows impulse response functions for an independent shock to each process, and the second shows a shock that is correlated according to the estimated system noise correlation. 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 — we might interpret this as reflecting their lives being more intertwined so they influence each other more. We visualise the independent shock type here, reflecting what our estimated model suggests should happen under an intervention to one or the other affect process.

```
print(tipredplots$Dynamics$Independent$TogetherZ)
```

Figure 20

Impulse Response Functions for Independent Shocks by Time Spent Together.



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

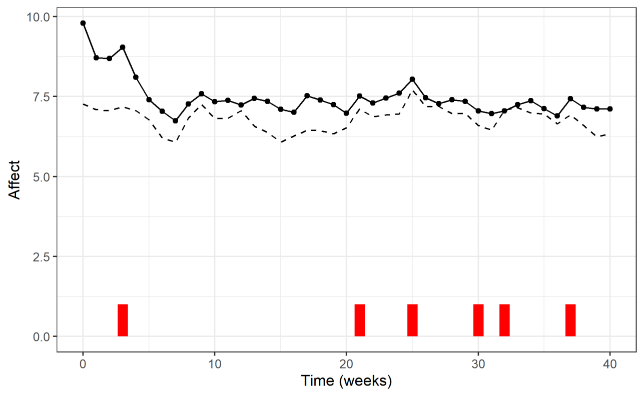
To generate data for a model with a time-dependent moderator of cross-effect state dependence, we first need to generate a time-dependent moderator that indicates when a couple is communicating (0 no communication, 1 = communication). To do this, we will sample a value of 0 or 1 from a binomial distribution (`Comm <- rbinom(n = NSubjects, size = 1, prob = .3)`) to reflect whether or not the couple was in contact over the preceding time period.

Second, we add the time-dependent moderator in our model of affect dynamics. This makes the rate of change for partner 1: `dAffect1 <- A*Affect1 + Across[subi] * Affect2 * Comm + B1[subi]`, where `Across` is multiplied by the time-dependent moderator `Comm` (which is either 0 or 1)at each time step. This ensures there is no cross-effect when the couple has not been communicating.

```
NSubjects <- 20
times <- seq(from=0, to=40, by=1) #time points of measurement
Nobs <- length(times) #number of observations per subject
t0Affect1 <- rnorm(n = NSubjects, mean = 5, sd = 2)
t0Affect2 <- rnorm(n = NSubjects, mean = 5, sd = 2)
A <- -.4 #continuous time state dependence
Across <- rnorm(n= NSubjects, mean = .1, sd=.05) #cross-effect

# generate continuous intercepts for each individual,
# assuming high affect individuals may partner with high affect
Bcommon <- rnorm(n = NSubjects, mean = 1, sd = .2)
B1 <- Bcommon + rnorm(n = NSubjects, mean = 2, sd = .3) #partner 1
B2 <- Bcommon + rnorm(n = NSubjects, mean = 2, sd = .3) #partner 2

# DIFFUSIONcov: covariance matrix for system noise
DIFFUSIONcov <- matrix(c(.1, .05, .05, .1), nrow=2, ncol=2)

#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))

Nsteps <- 100 #steps between each observation
row <- 0 #initialize row counter
for(subi in 1:NSubjects){
  for(obsi in 1:Nobs){ #for each observation of a subject
    row <- row + 1 #increment row counter
    Comm <- rbinom(n=1, size=1, prob=.3) #communication dummy
    if(obsi==1){ #set initial affect
      Affect1 <- t0Affect1[subi]
      Affect2 <- t0Affect2[subi]
    }
    if(obsi>1){ #update state via small steps in time
      for(stepi in 1:Nsteps){ #for each step
        #compute deterministic slopes
        dAffect1 <- A*Affect1+Across[subi]*Affect2*Comm + B1[subi]
        dAffect2 <- A*Affect2+Across[subi]*Affect1*Comm + B2[subi]
        # generate correlated noise scaled by stepsize
        systemNoise <- MASS::mvrnorm(n=1, mu=c(0,0),
          Sigma=DIFFUSIONcov/Nsteps)
        #update states using slope and time step, add noise
        Affect1 <- Affect1 + dAffect1/Nsteps + systemNoise[1]
        Affect2 <- Affect2 + dAffect2/Nsteps + systemNoise[2]
      }
    }
    data$Affect1[row] <- Affect1 #input affect data
    data$Affect2[row] <- Affect2 #input affect data
    data$Time[row] <- times[obsi] #input time data
    data$Comm[row] <- Comm #input communication dummy
    data$Subject[row] <- subi #input subject data
  }
}
# add measurement error
data$Affect1 <- data$Affect1 + rnorm(n=nrow(data),mean=0,sd = .05)
data$Affect2 <- data$Affect2 + rnorm(n=nrow(data),mean=0,sd = .05)
#plot trajectories with communication moderator
plot_trajectory(data[data$Subject==1,],y="Affect1",y2="Affect2")+
  geom_segment(aes(x=Time,xend=Time,y=0,yend=Comm,linewidth=.5),
  color="red")
```

Figure 21

Affect Trajectories for Both Partners in One Couple with Time-Dependent Communication Moderation.



Model Fitting

To specify a model with a time-dependent moderator in `ctsem` we deviate a little from the earlier example where the predictor had a direct impact on the process itself — here the predictor is only impacting the parameters of the system (the cross effect). First, we specify the name of our time-dependent predictor variable using `TDpredNames = c("Comm")`. Second, we define the main effect of our predictor using the `TDPREDEFFECT` argument. By default this is a freely estimated parameter. However, in our case we only want the moderation effect, so we set this to 0 (`TDPREDEFFECT = 0`). Third, we modify our drift matrix. Here, we are not going to exactly match our simulation, where we know for certain there is no cross-effect when the couple has a zero on the `Comm` predictor — we will also estimate a baseline parameter, partly serving as a verification of our simulation, and partly as a demonstration of possibilities. So in the `DRIFT` off diagonal elements we specify that our cross-effect is a combination of a baseline parameter (i.e., the cross effect when our moderator is 0) and an interaction term where the `Comm` predictor is directly referenced and multiplied by a parameter that represents the change in cross-effect when `Comm` is 1 — when the couple has been together for the past time period. So our cross-effect becomes `cross12base + cross12mod*Comm`. Sometimes when building more complex structures, it is important to think about where simplifications can be made – in this case, we reduce the number of parameters to be estimated by setting the auto and cross-effect parameters of each partner to be equal to each other, by using the same parameter names for both.

`ctsem` offers a lot of flexibility in terms of equations nested within each system matrix element. To make use of this flexibility, we need to explicitly tell `ctsem` which text strings represent parameters to be estimated in any `complex` equations — other text in the string is then treated as mathematical expressions (e.g., log, exp, +, -, *, /, etc.). For this we 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 `cross12base||TRUE`. Note that for the 'simple' kinds of parameters we have used up until now, we do not need to specify `PARS` at all — only when the parameters are nested within more complex equations do we need to specify `PARS` arguments.

```
m_discmod <- ctModel(
  manifestNames = c("Affect1",'Affect2'), #observed variables
  latentNames = c("Affect1",'Affect2'), #names of latent processes
  TDpredNames = c('Comm'), #time dependent predictors
  time = 'Time', #name of time column in dataset
  id = 'Subject', #name of subject column in dataset
  type='ct', #use continuous time
  MANIFESTVAR = c('residualSD1',0,
    0, 'residualSD2'), #sd's of the residual / measurement error
  LAMBDA = diag(1,2), #factor loading matrix
  MANIFESTMEANS=c(0,0), #no measurement intercept / offset
  CINT=c('B1||TRUE','B2||TRUE'), #cint with random effects
  TOMEANS=c('t0Affect1||TRUE','t0Affect2||TRUE'), #initial affect
  TDPREDEFFECT = 0, #disable main effect of comm predictor
  DRIFT = c( #cross effects now moderated and equal to each other
    'auto1', 'cross12base + cross12mod*Comm',
    'cross12base + cross12mod*Comm','auto1'),
  DIFFUSION = c( #system noise matrix
    'systemNoise1', 0, #sd for subj 1, 0 in upper triangle
    'systemNoiseCor', 'systemNoise2'),#correlation, sd for subj 2
    PARS=c('cross12base||TRUE','cross12mod||TRUE'))#extra params
```

```
f_discmod <- ctStanFit(datalong = data, ctstanmodel = m_discmod)
s_discmod <- summary(f_discmod)
print(s_discmod$popmeans)
```

```
                  mean     sd    2.5%     50%   97.5%
t0Affect1       5.4188 0.5506  4.3565  5.4035  6.4891
t0Affect2       5.1595 0.4290  4.3039  5.1675  5.9450
auto1          -0.4092 0.0134 -0.4371 -0.4087 -0.3848
systemNoise1    0.3181 0.0082  0.3031  0.3178  0.3347
systemNoiseCor  0.1183 0.0109  0.0972  0.1183  0.1386
systemNoise2   10.5587 0.1968 10.1723 10.5592 10.9607
residualSD1     0.0000 0.0000  0.0000  0.0000  0.0000
residualSD2     0.0648 0.0737  0.0062  0.0426  0.2452
B1              3.0365 0.1485  2.7626  3.0419  3.3280
B2              3.1341 0.1422  2.8531  3.1347  3.4190
cross12base    -0.0110 0.0143 -0.0394 -0.0104  0.0160
cross12mod      0.1021 0.0113  0.0792  0.1020  0.1240
```

Now we fit and summarise the results (in this case, the model may need to be compiled before fitting, which occurs automatically and usually takes a couple of minutes). The summary shows us that the `cross12base` parameter is estimated to be approximately 0, and the `cross12mod` parameter is estimated to be approximately 0.069031, 0.1643643, 0.1515376, 0.1074465, 0.0591798, 0.1790363, 0.0357792, 0.0693827, 0.0092141, 0.1216933, 0.1993079, 0.066702, 0.0837352, 0.1170703, 0.0541351, 0.1000825, 0.1065508, 0.1035336, 0.0411237, 0.0788486, as we specified in the simulation. This means that when the couple has been communicating for the past time period, the cross-effect is estimated to be approximately 0.15, there is influence from one to the other, otherwise there is no influence.

Different Types of Inputs — Nonlinear Change with a Persistent External Shock (Continuous Time)

The final approach we will consider in terms of model structure is how we can add additional processes to the system that are not directly observed, but can be used to structure the system dynamics. For this example we will turn back to the example where we used therapy as an input to the system, but rather than assume therapy impacts the system and the effect transmits through time with the same dynamics as the rest of the system, we will allow for the persistence, or 'shape' of the effect to differ from the rest of the system. This approach can be generalized to achieve things such as separated trends and dynamics processes (Driver & Tomasik, 2023), cross-effects that operate at different time scales (Haehner et al., 2025), oscillatory or more complex dynamic patterns (Voelkle & Oud, 2013), and more.

Our original model implies that the therapy effect is immediate and persists according to the system dynamics. This is probably the simplest approach to thinking about such effects, but in reality, the effects of an intervention or event 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.

To demonstrate this approach, instead of relying on the more 'general' system parameters to capture the long term trajectory in response to couples therapy sessions — remembering that from the beginning of this tutorial we have always worked with a rising affect over time — we can now explicitly model this rise, and maintenance at a new level, as a function of therapy sessions For more examples of intervention effects and how to specify them in `ctsem` we refer the reader to (Driver & Voelkle, 2018b). A major benefit of this approach is that, for example, patients may have very different intervention schedules — allowing the trend to develop based on the specific schedule, rather than a general assumption of a specific form, can be more realistic and flexible.

To do this, we need to specify a latent process which is impacted by our observed intervention variable, and is *not measured by any variables*. This process is used to represent the hypothetical development of the intervention effect over time. To try to make this explicit:

Simple impulse form: Intervention directly impacts affect process, dissipation governed by system dynamics.

Separated form: Intervention affects a separate latent process that is independent of the main system dynamics, with either estimated or fixed dynamic parameters — this process in turn then impacts the affect process.

This latent intervention process can have its own autoregressive parameter which allows us to specify an accelerating, dissipating, or stable effect across time. That is, if the state dependence is set to 0, our latent intervention process will remain at a stable value (e.g., remaining at 1 after the initial upwards shock) and thereby generate a permanent shift over time. If the state dependence is set to a negative value (e.g. -0.1), then the effect of the intervention will dissipate over time, but not as quickly as it would have in the pure impulse form.

Simulation

In our data generating block, we include an input variable `M` which directly influences the state of our latent intervention process `TherapyAcc` (accumulated therapy), which is included in much the same way as additional latent processes were previously simulated. The rate of change in our latent intervention process `dTherapyAcc` is given by the previous value of `TherapyAcc` multiplied by a state dependence coefficient, `dTherapyAcc <- AM*TherapyAcc`. Because the state dependence coefficient is 0 (autoregression = 1) in the case of a persistent input effect, the rate of change is always 0. For simplicity, at the time of therapy sessions the input effect `M` is added directly to `TherapyAcc` — there is no need to track this via the rate of change, as it is an instantaneous impulse.

Between therapy sessions, because the state dependence parameter is zero, the state of the latent intervention process `TherapyAcc` will not change; the therapy sessions generate a step-like process that reflects a persistent input process, gradually accumulating over time as therapy sessions occur (one could argue that it should probably also slowly decay over time, but we will leave that for now). We nevertheless show the code for updating the latent intervention process with a small step in time, to illustrate the process.

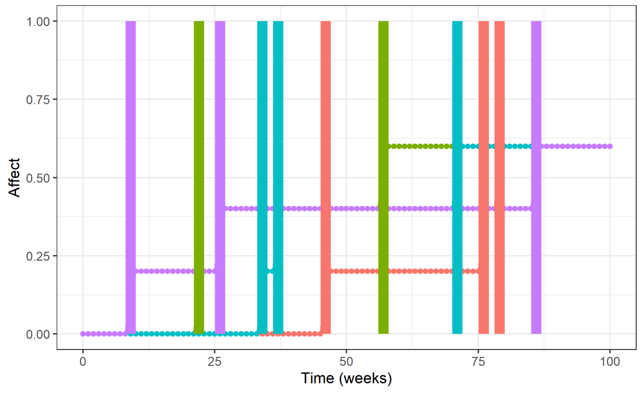
To model the impact of our latent intervention process on our affect process, we make the rate of change in affect dependent on the state of the intervention process at the current moment (`dAffect <- A*Affect + B + TherapyAcc`) — this is just another cross-effect in our drift matrix, from the matrix equation perspective.

```r
NSubjects <- 20
times <- seq(from=0, to=100, by=1) #time points of measurement
Nobs <- length(times) #number of observations per subject
t0Affect <- rnorm(n = NSubjects, mean = 5, sd = 2)
A <- -.1 #continuous time state dependence
B <- 0.5 #continuous intercept reduced, growth via therapy effect
G <- .2 #system noise coefficient
M <- 0.2 #input variable
AM <- 0 #latent input process autoregression

# randomly sample 3 time points for therapy for each subject
therapyTimes <- lapply(1:NSubjects, function(x){
  return(sample(times, size = 3, replace = FALSE))
})

#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))
```

```r
Nsteps <- 100 # steps between each observation
row <- 0 #init current row tracker
for(subi in 1:NSubjects){#for each subject
  for(obsi in 1:Nobs){ #for each observation of a subject
    row <- row + 1 # increment row counter
    #create therapy dummy:
    Therapy <- as.integer(times[obsi] %in% therapyTimes[[subi]])
    if(obsi==1){ #if first time point
      Affect <- t0Affect[subi] #set initial affect
      TherapyAcc <- 0 # initialize latent input process
    }
    if(obsi>1){ #otherwise update state via small steps in time
      for(stepi in 1:Nsteps){ #for each step
        # compute deterministic slopes
        dTherapyAcc <- AM*TherapyAcc # therapy slope update
        dAffect <- A*Affect+B+TherapyAcc #affect slope update
        #update states using slope and time step, add noise
        TherapyAcc <- TherapyAcc + dTherapyAcc/Nsteps
        Affect <- Affect + dAffect/Nsteps +
         G * rnorm(n=1, mean=0, sd=sqrt(1/Nsteps)) #system noise
      }
    }
    TherapyAcc <- TherapyAcc + M*Therapy # effect of therapy
    #store data:
    data$Affect[row] <- Affect #input affect data
    data$Time[row] <- times[obsi] #input time data
    data$Subject[row] <- subi #input subject data
    data$Therapy[row] <- Therapy #input therapy dummy
    data$TherapyAcc[row] <- TherapyAcc #input intervention process
  }
}
#add measurement error
data$Affect<-data$Affect+rnorm(n=nrow(data),mean=0,sd =.05)
# plot therapy accumulation process and therapy dummies
plot_trajectory(data[data$Subject %in% 1:4, ],
 color_var = "as.factor(Subject)", y='TherapyAcc') +
 geom_segment(aes(x=Time,xend=Time,y=0,yend=Therapy,linewidth=.5))
```

Figure 22

Latent Intervention Process (TherapyAcc) for Four Subjects with Therapy Sessions.



The plot shows the latent intervention process `TherapyAcc` for the first four subjects in our dataset. We can see that at each therapy session (indicated by the vertical lines), the latent intervention process jumps up by 0.2 (the value of `M`), and then remains stable until the next therapy session.

Model & Fitting

We want to specify a latent intervention process that only reflects the time course 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. We have not had to deal with the `TOVAR` argument before — it is only relevant when there are no individual differences in the starting value of the process, which occurs either when the differences are disabled, or when the starting value is fixed, as in this case. The `TOVAR` matrix provides variance at the initial time point for each subject. We do not want this variance, so we can specify a single zero (which will actually create an appropriate size matrix of zeros). Finally, we fix the therapy processes continuous intercept `CINT` to zero since there is no constant growth in the latent intervention process.

Next, we set the cross-effect of our latent intervention process to 1 using the `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 = c(0,"TherapyEffect")`). Thus, our observed intervention directly impact the latent intervention process, which in turn directly impacts our latent affect process. Instead of estimating the effect of Therapy on TherapyAcc, alternatively we could have fixed this to 1 and estimated the effect of TherapyAcc on Affect.

```r
m_persist <- ctModel(
  manifestNames = "Affect", #observed variables
  latentNames = c("Affect","TherapyAcc"), #latent process names
  TDpredNames = 'Therapy', #time dependent predictors
  time = 'Time', #time column
  id = 'Subject', #subject column
  type='ct', #continuous time
  MANIFESTVAR = 'residualSD', #residual variance
  LAMBDA = matrix(1,nrow=1,ncol=2), #factor loading matrix
  MANIFESTMEANS=0, #no measurement intercept
  CINT=c('B||FALSE', #cint with *no* random effects for Affect,
    "0||FALSE"), # and no CINT at all for therapy accumulation
  TDPREDEFFECT = c(0,"TherapyEffect"), #Therapy on TherapyAcc
  TOMEANS=c("t0Affect||TRUE", #initial affect with random effects,
    "0||FALSE"), # and therapy accumulation starts at 0
  TOVAR=0, # no within-subject variance in starting values
  DRIFT = c( #auto effect and TherapyAcc effect on affect.
    'stateDependence', 1, #autoeffect
    0, 0), #therapy accumulation process is static
  DIFFUSION = c('systemNoise',0, #system noise for affect
    0, 0)) #no system noise for therapy accumulation
```

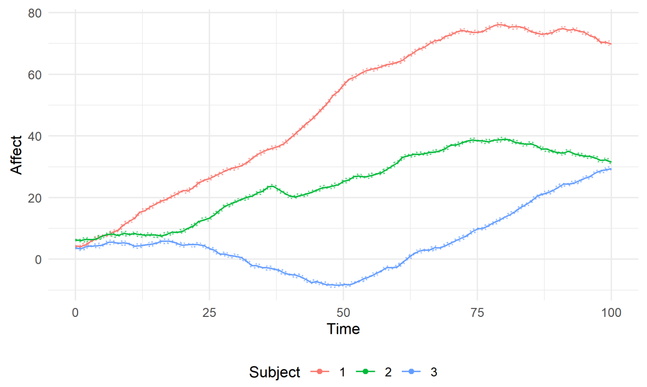
```r
f_persist <- ctStanFit(datalong = data, ctstanmodel = m_persist)
```

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 increase in affect at the time of the intervention, with the effect persisting over time. Note here that the effect is not immediate, but accumulates and persists in comparison to the earlier example.

```r
ctKalman(f_persist, plot=TRUE, subjects=1:3, kalmanvec='ysmooth')
```

Figure 23

Smoothed Affect Estimates for Three Subjects with Persistent Therapy Input.



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, and we cannot cover all aspects and nuance here. To get people started with model evaluation, we will demonstrate some tools and approaches 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.

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 trajectory. If we have data from 20 patients over 21 weeks, we can use the R^2 estimate to check how much variance in the data has been explained by the model. For our example we will generate data from a known model (simplifying the explanations) but normally the true model is not known! .

Generating Data

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

```r
NSubjects <- 20
times <- seq(from=0, to=20, by=1) #time points of measurement
Nobs <- length(times) #number of observations per subject
```

```
t0Affect <- 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 #steps between each observation
row <- 0 #init current row tracker
for(subi in 1:NSubjects){
  for(obsi in 1:Nobs){ #for each observation of a subject
    row <- row + 1
    #if first time point, set to initial affect
    if(obsi==1) Affect <- t0Affect[subi]
    if(obsi>1){ #else compute new state via small steps in time
      for(stepi in 1:Nsteps){ #every step, do...
        dAffect <- A*Affect + B #compute affect slope
        Affect <- Affect + #new state = old state +
        dAffect * 1/Nsteps + #deterministic change +
          G * rnorm(n=1, mean=0, sd=sqrt(1/Nsteps)) #random change
      }
    }
    data$Affect[row] <- Affect #input affect data
    data$Time[row] <- times[obsi] #input time data
    data$Subject[row] <- subi #input subject data
  }
}
#add measurement error
data$Affect <- data$Affect+rnorm(n=nrow(data), mean = 0, sd = .05)
```

Fitting the Wrong Model: Linear Growth

We now specify and fit our linear growth model, with no state-dependency and no system noise, in `ctsem`.

```
m_linear <- ctModel( #define the ctsem model
  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', #continuous time
  MANIFESTVAR = 'residualSD', #sd of residual / measurement error
  LAMBDA = matrix(1,nrow=1,ncol=1), #factor loading matrix
  MANIFESTMEANS=0, #no measurement intercept
  CINT='B||FALSE', #continuous intercept with *no* random effects
  T0MEANS='t0Affect||TRUE', #initial affect with random effects
  DRIFT = 0,
  DIFFUSION = 0)
```

```
f_linear <- ctStanFit(datalong = data, ctstanmodel = m_linear)
```

A common form of model evaluation is to consider how much of the variance in the data has been explained by the model. This is typically quantified using the R^2 statistic (Nakagawa & Schielzeth, 2013) or some variation thereof. 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) and subtracting this from 1. Because in `ctsem` models the total variance is not necessarily constant across individuals or time, this is computed using standardized residuals, and reported (as standardized residual variance / covariance) at the top of the ctsem summary output.

```
s=summary(f_linear) #store summary of the fit
print(s$residCovStd) #print standardized residual (co)variance
```

```
       Affect
Affect   0.22
```

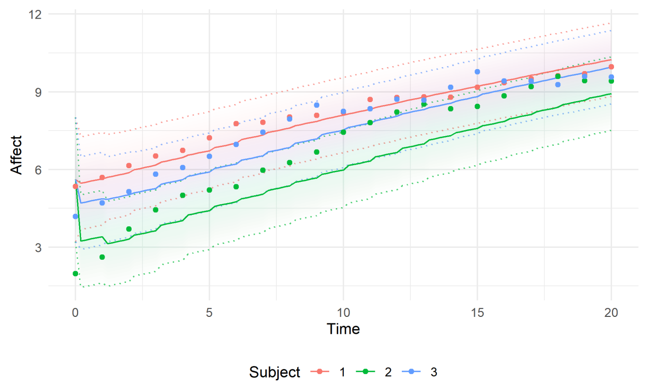
Now we can use the standardized residual variance estimate (reported at the top of summary under 'residCovStd', equivalent to 1 - R^2) to see how much of the variance in our empirical data our model has accounted for. We can see that our model explains a high proportion of the variance in our observations. This is fairly typical for repeated measures data, and doesn't imply that our model is appropriate, but provides a useful baseline for comparison.

Let's begin considering model appropriateness by looking at the models forward predictions (i.e., conditional on parameter estimates and earlier data, represented by lines in the plot) against the observed data (represented by dots) for a few subjects:

```
ctKalman(f_linear,plot=TRUE,subjects=1:3,
 kalmanvec = c('y','yprior')) #y=observed, yprior=predictions
```

Figure 24

Predictions from Misspecified Linear Model Versus Observed Data for Three Subjects.



A first question that might arise here, is why aren't the model predictions the simple linear trajectory we specified? This occurs because as the model steps through time and 'sees' more data, it learns the individual difference parameters for each subject, and thus the predictions adjust and do not follow the simple linear trajectory. If we plotted the 'smoothed' predictions (conditional on all data, not just earlier data, use `ysmooth` instead of `yprior`) they would follow the linear trajectory. The forward predictions we see here, conditional only on parameter estimates and past data, are what the likelihood calculations and residuals are based on. If using an appropriate model, the residuals here — the difference between the observed and predicted data — should be entirely *unpredictable* based on earlier data or residuals.

While far from a formal test, a visual inspection of the model predictions versus observed data suggests that this is *not* the case, as we can see some clear structure in the observed data that is not being captured by the model. This is visible in that when the model prediction is too low at one observation, it is also likely to be too low at the next observation

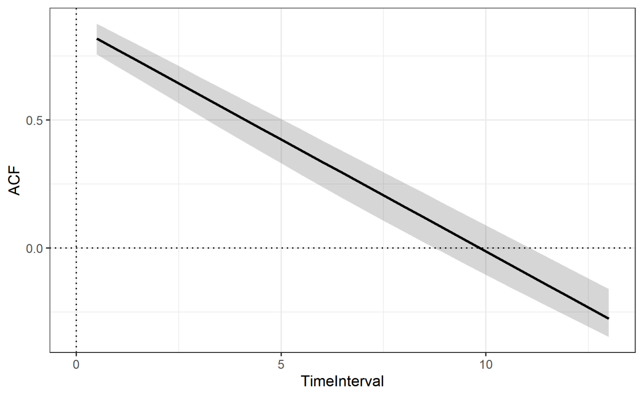
— our residuals are *autocorrelated*. Autocorrelated residuals indicate that our model leaves some predictable structure unexplained (Gelman et al., 1996) and we could modify our model to improve the predictions. This state of affairs is broadly referred to as *model misspecification*. The predictions may still be useful for some purposes, but we should be extremely careful about interpretations of the model parameters or other generalisations.

An approach to examine all residuals at once in a more comprehensive manner is to check their autocorrelation. 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(f_linear) #plot autocorrelation of residuals
```

Figure 25

Autocorrelation of Residuals from the Misspecified Linear Model.

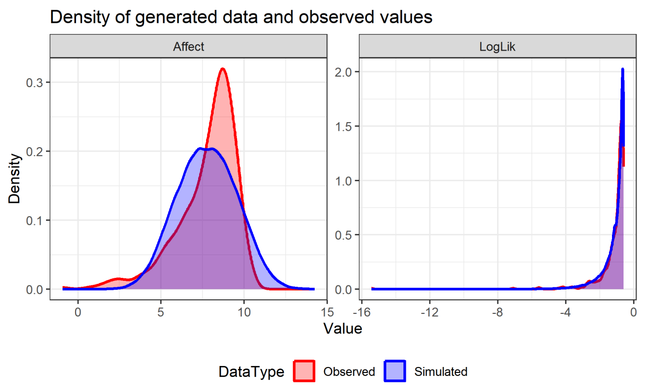


From the plot we can see that there is a high autocorrelation between residuals, about 0.6 between temporally adjacent residuals. So now we know two things: (1) Our model explains a substantial amount of the variance in our data, (2) there is still room for improvement. The natural follow-up question 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. This approach is analogous to posterior predictive checks in a Bayesian framework, but can be applied in non-Bayesian contexts as well (Gelman & Shalizi, 2013).

```
f_linear <- ctStanGenerateFromFit(f_linear, nsamples=200,
  fullposterior = F, cores = 2) #add generated data to fit object
postpred<-ctPostPredPlots(f_linear) #create plots
print(postpred[[1]]) #show first plot --- overall density
```

Figure 26

Posterior Predictive Check: Density of Likelihood and Variables (Linear Misspecification).

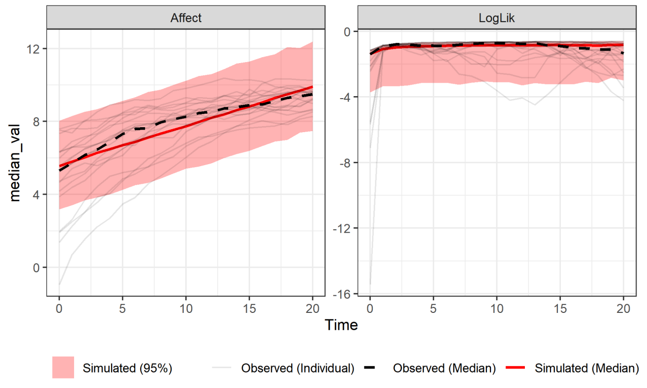


These first plots show the overall density plots of row-wise likelihood and variables. It is difficult to get a clear sense of exactly where things are imperfect from such a plot, but this aggregate view provides another approach to see that there clearly are some patterns in the data that deviate from the model's expectations. Perfect overlap between the empirical and model-implied distributions for each variable doesn't necessarily mean the model is appropriate, but does at least indicate that the overall distributions are similar.

```
print(postpred[[2]]) #show second plot --- expectations over time
```

Figure 27

Posterior Predictive Check: Model-Implied Versus Empirical Expectations over Time (Linear Misspec).



These second plots show the model implied expectations (red) versus the empirical (black). Here it is quite clear that we are fitting a linear model to a nonlinear trajectory. Our model also seems to underestimate the variance at earlier time points while overestimating the variance at later observations. But how can we improve our model to better account for these observations?

Taking a closer look at the observed individual growth trajectories (fainter black lines), 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 to capture this relationship.

Fitting a Better Model: Adding State-Dependence in Continuous Time

```
m_statedep <- ctModel(
  manifestNames = "Affect", #observed variables
  latentNames = "Affect", #latent processes
  time = 'Time', #time column
  id = 'Subject', #subject column
  type='ct', #continuous time
  MANIFESTVAR = 'residualSD', #residual variance
  LAMBDA = matrix(1,nrow=1,ncol=1), #factor loading matrix
  MANIFESTMEANS=0, #no measurement intercept
  CINT='B||FALSE', #continuous intercept with *no* random effects
  T0MEANS='t0Affect||TRUE', #initial affect with random effects
  DRIFT = 'stateDependence', #state-dependence
  DIFFUSION = 0) #no system noise
```

```
f_statedep <- ctStanFit(datalong = data, ctstanmodel = m_statedep)
```

```
s=summary(f_statedep) #store summary of the fit
print(s$residCovStd) #print standardized residual (co)variance
```

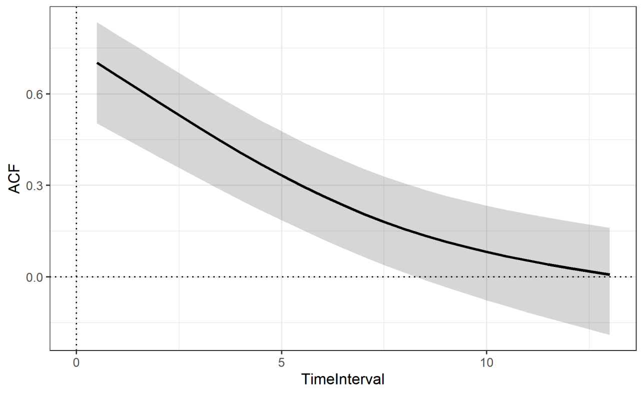
```
       Affect
Affect  0.098
```

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(f_statedep) #plot autocorrelation of residuals
```

Figure 28

Autocorrelation of Residuals from the State-Dependence Model.

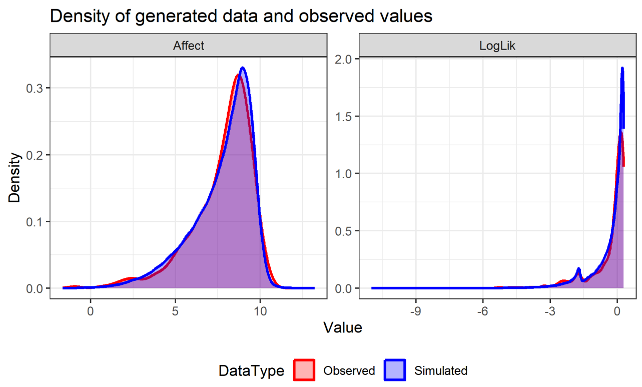


We can see that the autocorrelation has also dropped substantially, but there is still a moderate amount of autocorrelation in our residuals. Which is rightly telling us that there is more predictable variation that our model can leverage. So let's try to find out what it is by plotting the model generated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

```
fit <- ctStanGenerateFromFit(f_statedep, nsamples=200,
  fullposterior = F, cores = 2) #add generated data to fit object
postpred <- ctPostPredPlots(f_statedep) #create plots
print(postpred[[1]]) #show first plot --- overall density
```

Figure 29

Posterior Predictive Check: Density of Likelihood and Variables (State-Dependence Model).

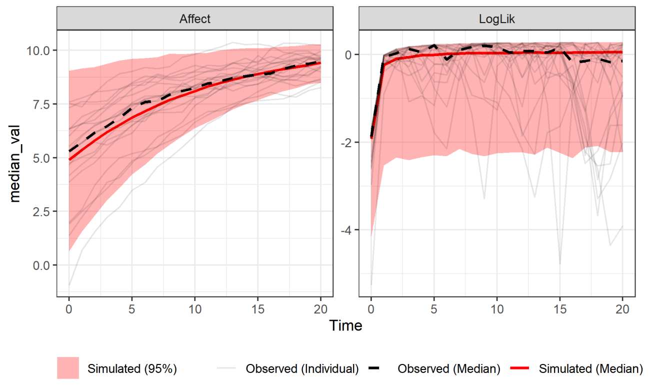


Looking at the first plot, the model implied distributions are now quite similar to the empirical distributions, but there is still some difference — it's not so easy to determine if such a difference is important, from this visual.

```
print(postpred[[2]]) #show second plot --- expectations over time
```

Figure 30

Posterior Predictive Check: Model-Implied Versus Empirical Expectations over Time (State-Dependence Model).



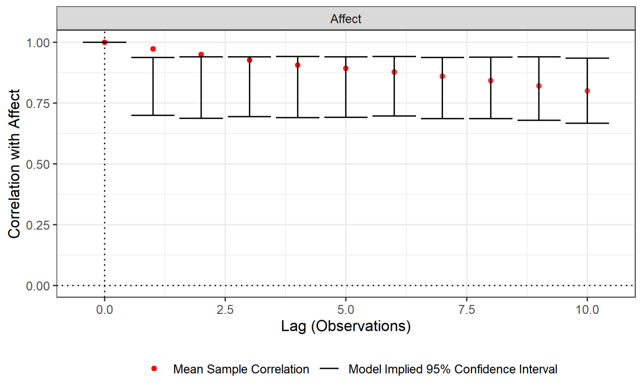
The trajectory plot shows that our model implied trajectory almost perfectly fits our observed trajectory, and the variance now also reduces over time in line with the observed trajectories converging to a similar level. So, we know there appears to be some predictable variation that our model could better account for, but it's difficult to determine how to do this. Examining empirical versus model implied lagged correlations can provide more insight:

```
ctFitCovCheck(f_statedep,plot = TRUE,cor=TRUE)
```

```
$Affect
```

Figure 31

Empirical Versus Model-Implied Lagged Correlations (State-Dependence Model).



This plot shows the empirical versus model implied lagged correlations for each variable (in a multivariate scenario, we would also see cross-correlations). We can see that the model implied correlations between one time point and later time points are stable, while the empirical correlations decrease over time. This makes sense because although we've now specified a state dependence parameter in the model, causing growth to slow and stabilise over time, we didn't allow for any *new* sources of variation to be introduced into the process — the same variation that was there in the beginning persists on and on, smaller in magnitude, but equally correlated with earlier time points. Introducing a system noise term to our model will allow for new sources of variation to be introduced into the process, and we can see how this changes the lagged correlations.

```
m_sysnoise <- ctModel(
  manifestNames = "Affect", #observed variables
  latentNames = "Affect", #latent processes
  time = 'Time', #time column
  id = 'Subject', #subject column
  type='ct', #continuous time
  MANIFESTVAR = 'residualSD', #residual variance
  LAMBDA = matrix(1,nrow=1,ncol=1), #factor loading matrix
  MANIFESTMEANS=0, #no measurement intercept
  CINT='B||FALSE', #continuous intercept with *no* random effects
  TOMEANS='t0Affect||TRUE', #initial affect with random effects
  DRIFT = 'stateDependence', #state-dependence
  DIFFUSION = 'systemNoise') #system noise
```

```
f_sysnoise <- ctStanFit(datalong = data, ctstanmodel = m_sysnoise)
```

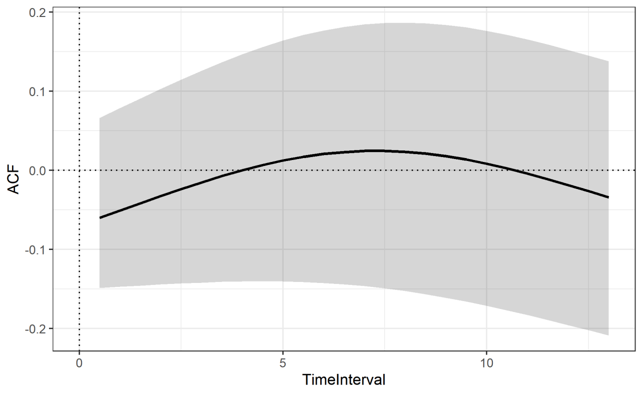
```
s=summary(f_sysnoise) #store summary of the fit
print(s$residCovStd) #print standardized residual (co)variance
```

```
       Affect
Affect  0.081
```

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

Figure 32

Autocorrelation of Residuals from the Model with System Noise.



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 lagged correlations that map well onto the observed lagged correlations.

In empirical contexts, this process is going to be more complicated — individual differences will likely need more serious consideration, and very often it will be difficult to impossible to obtain a model that appears perfect while also retaining some level of interpretability. Sensitivity checks, wherein core inferences are checked across a range of model specifications, can help to assess the robustness of the inferences to different model specifications (for an example of the approach in the context of ctsem see Driver, 2025).

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 et al., 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 et al., 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). 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 & Willoughby, 2017; Hamaker et al., 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 et al., 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 et al., 2011), and how short and long term timescales may be considered in the one model (Boker et al., 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 & 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 & 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 & 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 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 exploration, fit, and interpretation of complex model structures, with flexibility to approximate many different theoretical structures of human dynamic systems.

References

Aalen, O. O., Røysland, K., Gran, J. M., Kouyos, R., & Lange, T. (2016). Can we believe the DAGs? A comment on the relationship between causal DAGs and mechanisms. *Statistical Methods in Medical Research*, *25*(5), 2294–2314. https://doi.org/10.1177/0962280213520436

Asparouhov, T., Hamaker, E. L., & Muthén, B. (2018). Dynamic Structural Equation Models. *Structural Equation Modeling: A Multidisciplinary Journal*, *25*(3), 359–388. https://doi.org/10.1080/10705511.2017.1406803

Bates, D., Mächler, M., Bolker, B., & Walker, S. (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, D., & Willoughby, M. T. (2017). On the practical interpretability of cross-lagged panel models: Rethinking a developmental workhorse. *Child Development*, *88*(4), 1186–1206. https://doi.org/10.1111/cdev.12660

Boker, S. M., Daniel, K. E., & Orzek, J. (2024). Separating Long-Term Equilibrium Adaptation from Short-Term Self-Regulation Dynamics Using Latent Differential Equations. *Multivariate Behavioral Research*, *59*(6), 1177–1187. https://doi.org/10.1080/00273171.2023.2228302

Borsboom, D., & Haslbeck, J. (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, D., van der Maas, H. L. J., Dalege, J., Kievit, R. A., & Haig, B. D. (2021). Theory construction methodology: A practical framework for building theories in psychology. *Perspectives on Psychological Science*, *16*(4), 756–766. https://doi.org/10.1177/1745691620969647

Butler, E. A., & Randall, A. K. (2013). Emotional coregulation in close relationships. *Emotion Review*, *5*(2), 202–210. https://doi.org/10.1177/1754073912451630

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

Curran, P. J., Obeidat, K., & Losardo, D. (2010). Twelve frequently asked questions about growth curve modeling. *Journal of Cognition and Development*, *11*(2), 121–136. https://doi.org/10.1080/15248371003699969

Driver, C. C. (2023). Inference with cross-lagged effects – common causes in dynamic systems. *PsyArXiv*. https://doi.org/10.31234/osf.io/y3e9d

Driver, C. C. (2025). Inference with cross-lagged effects—Problems in time. *Psychological Methods*, *30*(1), 174–202. https://doi.org/10.1037/met0000665

Driver, C. C., Oud, J. H. L., & Voelkle, M. C. (2017). Continuous Time Structural Equation Modeling with R package ctsem. *Journal of Statistical Software*, *77*(5), 1–35. https://doi.org/10.18637/jss.v077.i05

Driver, C. C., & Tomasik, M. J. (2023). Formalizing developmental phenomena as continuous-time systems: Relations between mathematics and language development. *Child Development*, *94*(6), 1454–1471. https://doi.org/10.1111/cdev.13990

Driver, C. C., & Voelkle, M. C. (2018a). Hierarchical Bayesian continuous time dynamic modeling. *Psychological Methods*, *23*(4), 774–799. https://doi.org/10.1037/met0000168

Driver, C. C., & Voelkle, M. C. (2018b). Understanding the time course of interventions with continuous time dynamic models. In K. van Montfort, J. H. L. Oud, & M. C. Voelkle (Eds.), *Continuous time modeling in the behavioral and related sciences* (pp. 79–109). Springer International Publishing. //www.springer.com/de/book/9783319772189

Driver, C. C., Voelkle, M. C., & Oud, J. H. L. (2025). *Ctsem: Continuous time structural equation modelling* [Manual]. CRAN. https://cran.r-project.org/package=ctsem

Durbin, J., & Koopman, S. J. (2012). *Time series analysis by state space methods* (2nd ed.). Oxford University Press.

Epskamp, S. (2020). Psychometric network models from time-series and panel data. *Psychometrika*, *85*(1), 206–231. https://doi.org/10.1007/s11336-020-09697-3

Gabry, J., Simpson, D., Vehtari, A., Betancourt, M., & Gelman, A. (2019). Visualization in bayesian workflow. *Journal of the Royal Statistical Society: Series A*, *182*(2), 389–402. https://doi.org/10.1111/rssa.12378

Gelman, A., Meng, X.-L., & Stern, H. (1996). Posterior predictive assessment of model fitness via realized discrepancies. *Statistica Sinica*, *6*(4), 733–760. https://www.jstor.org/stable/24306036

Gelman, A., & Shalizi, C. R. (2013). Philosophy and the practice of Bayesian statistics. *British Journal of Mathematical and Statistical Psychology*, *66*(1), 8–38. https://doi.org/10.1111/j.2044-8317.2011.02037.x

Haehner, P., Driver, C. C., Hopwood, C. J., Luhmann, M., Fliedner, K., & Bleidorn, W. (2025). The dynamics of self-esteem and depressive symptoms across days, months, and years. *Journal of Personality and Social Psychology*. https://doi.org/10.1037/pspp0000542

Hamaker, E. L., Kuiper, R. M., & Grasman, R. P. P. P. (2015). A critique of the cross-lagged panel model. *Psychological Methods*, *20*(1), 102–116. https://doi.org/10.1037/a0038889

Harvey, A. C. (1989). *Forecasting, structural time series models and the kalman filter*. Cambridge University Press.

Haslbeck, J. M. B., Ryan, O., Robinaugh, D. J., Waldorp, L. J., & Borsboom, D. (2022). Modeling psychopathology: From data models to formal theories. *Psychological Methods*, *27*(6), 930–957. https://doi.org/10.1037/met0000303

Higham, D. J. (2001). An algorithmic introduction to numerical simulation of stochastic differential equations. *SIAM Review*, *43*(3), 525–546. https://doi.org/10.1137/S0036144500378302

Kalman, R. 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, D. 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, D. A., Kashy, D. A., & Cook, W. L. (2006). *Dyadic data analysis*. Guilford Press.

Kloeden, P. E., & Platen, E. (1992). *Numerical solution of stochastic differential equations*. Springer.

Kuiper, R. M., & Ryan, O. (2018). Drawing conclusions from cross-lagged relationships: Re-considering the role of the time-interval. *Structural Equation Modeling: A Multidisciplinary Journal*, *25*(5), 809–823. https://doi.org/10.1080/10705511.2018.1431046

Lütkepohl, H. (2005). *New introduction to multiple time series analysis*. Springer.

Nakagawa, S., & Schielzeth, H. (2013). A general and simple method for obtaining R^2 from generalized linear mixed-effects models. *Methods in Ecology and Evolution*, *4*(2), 133–142. https://doi.org/10.1111/j.2041-210X.2012.00261.x

Oravecz, Z., Tuerlinckx, F., & Vandekerckhove, J. (2011). A hierarchical latent stochastic differential equation model for affective dynamics. *Psychological Methods*, *16*(4), 468–490. https://doi.org/10.1037/a0024375

Ou, L., Hunter, M. D., & Chow, S.-M. (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, D. J., Haslbeck, J. M. B., Ryan, O., Fried, E. I., & Waldorp, L. J. (2021). Invisible hands and fine calipers: A call to use formal theory as a toolkit for theory construction. *Perspectives on Psychological Science*, *16*(4), 725–743. https://doi.org/10.1177/1745691620974697

Rohrer, J. M., & Murayama, K. (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, O., & Hamaker, E. L. (2022). Time to intervene: A continuous-time approach to network analysis and centrality. *Psychometrika*, *87*(1), 214–252. https://doi.org/10.1007/s11336-021-09767-0

van Geert, P. (2011). The contribution of complex dynamic systems to development. *Child Development Perspectives*, *5*(4), 273–278. https://doi.org/10.1111/j.1750-8606.2011.00197.x

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

Voelkle, M. C., Oud, J. H. L., Davidov, E., & Schmidt, P. (2012). An SEM approach to continuous time modeling of panel data: Relating authoritarianism and anomia. *Psychological Methods*, *17*(2), 176–192. https://doi.org/10.1037/a0027543

Wickham, H. (2011). Ggplot2. *Wiley Interdisciplinary Reviews: Computational Statistics*, *3*(2), 180–185.

Wilson, R. C., & Collins, A. G. (2019). Ten simple rules for the computational modeling of behavioral data. *eLife*, *8*, e49547. https://doi.org/10.7554/eLife.49547