Tutorial: Fitting Dynamic Tree-Based Item Response Models to Intensive Polytomous Time Series Eye-Tracking Data Using R

Matthew Naveiras and Sun-Joo Cho, Vanderbilt University

4/26/2020

Funding Source: This work was supported by the National Science Foundation (SES1851690, https://www.nsf.gov/awardsearch/showAward?AWD_ID=1851690).

Introduction

This website provides a tutorial for fitting dynamic tree-based item response (IRTree) models as presented in Cho, Brown-Schmidt, De Boeck, & Shen (in press, https://link.springer.com/article/10.1007/s11336-020-09694-6?) using R. In this tutorial we will show:

- how to recode your data for the application of dynamic IRTree models (Data Management),
- how to calculate and plot the autocorrelations, partial autocorrelations, and trend of your data (Data Description),
- how to specify dynamic IRTree models and estimate the models (Model Specification and Estimation),
- how to compare different dynamic IRTree models to select the best-fitting model regarding random effects (Model Selection), and
- how to evaluate the adequacy of the selected dynamic IRTree model (Model Evaluation).

Data Management

This section will show how to recode your data to include variables for node values (*node*) and recoded responses (*yy*).

First, read the file for your data (as a data frame) into R:

```
setwd("C:\\Users\\MyName\\Documents\\Code")
data <- read.table("data.txt",sep="\t",header=TRUE)
```

Here is what your data should (roughly) resemble:

```
##   trial item person time contrast privileged y
## 1   273    5      1  180      0.5        0.5 3
## 2   273    5      1  190      0.5        0.5 3
## 3   273    5      1  200      0.5        0.5 3
## 4   273    5      1  210      0.5        0.5 3
## 5   273    5      1  220      0.5        0.5 3
## 6   273    5      1  230      0.5        0.5 3
## 7   273    5      1  240      0.5        0.5 1
## 8   273    5      1  250      0.5        0.5 1
```

The only variable we need for data management is the response data (y).

Second, we'll duplicate the data based on the number of nodes. Each duplicate will have that node's value for the *node* variable. In this example, we will have 2 nodes, therefore, the final data frame *data.node* will be a data frame composed of 2 copies of the original data, with the first copy having $node = 1$, and the second copy having $node = 2$.

```
number.of.nodes <- 2
for (node in 1:number.of.nodes){
data.copy <- data
data.copy$node <- rep(node, nrow(data))
if (node==1){data.node <- data.copy} else {data.node <- rbind(data.node,data.copy)}
}
```

Line 1 in the above code sets the number of nodes (in this example 2), and the loop in lines 2-6 creates the new data frame with the inclusion of the *node* variable, first by copying the data for each node and then combining the copies into a single data frame. Note that the number of rows for *data.node* [$nrow(data.node)$] should be equal to $number.of.nodes \times nrow(data)$.

Third, the following table illustrates how the recoded responses yy can be created given the values of *node* and y at each observation:

Response	*node*	y	yy
target	1	1	1
competitor	1	2	1
unrelated objects	1	3	0
target	2	1	1
competitor	2	2	0
unrelated objects	2	3	NA

The above table is translated into code as *yy.list* below:

```
yy.list <- list(list(1,1,0),list(1,0,NA))
```

Notice that the value for yy given a value for *node* and a value for y is indexed as $yy.list[[node]][[y]]$. For example, if $node = 1$ and $y = 2$, then $yy = yy.list[[1]][[2]] = 1$, and if $node = 2$ and $y = 3$, then $yy = yy.list[[2]][[3]] = NA$.

Fourth, the following adds yy as a variable to our data *data.node*:

```
for (observation in 1:nrow(data.node)){
data.node$yy[observation] <- yy.list[[data.node$node[observation]]][[data.node$y[observation]]]
}
```

Your *data.node* should now include values for *node* (ranging from 1 to the number of nodes) and yy:

```
##   trial item person time contrast privileged y node yy
## 1   273    5      1  180      0.5        0.5 3    1  0
## 2   273    5      1  190      0.5        0.5 3    1  0
## 3   273    5      1  200      0.5        0.5 3    1  0
## 4   273    5      1  210      0.5        0.5 3    1  0
## 5   273    5      1  220      0.5        0.5 3    1  0
## 6   273    5      1  230      0.5        0.5 3    1  0
## 7   273    5      1  240      0.5        0.5 1    1  1
## 8   273    5      1  250      0.5        0.5 1    1  1
```

Data Description

This section will show how to calculate empirical logits ($logit = log(\frac{proportion}{1-proportion})$) for your data, and then use those empirical logits to explore change processes (autocorrelation, partial autocorrelation, and trend). We'll then use those calculations to create plots for the autocorrelations, partial autocorrelations, and trend using the ggplot2 package (Wickham, 2016).

Calculation: Empirical logit

We'll be using the data set we created in the data description step (*data.node*):

```
##   trial item person time contrast privileged y node yy
## 1   273    5      1  180      0.5        0.5 3    1  0
## 2   273    5      1  190      0.5        0.5 3    1  0
## 3   273    5      1  200      0.5        0.5 3    1  0
## 4   273    5      1  210      0.5        0.5 3    1  0
## 5   273    5      1  220      0.5        0.5 3    1  0
## 6   273    5      1  230      0.5        0.5 3    1  0
## 7   273    5      1  240      0.5        0.5 1    1  1
## 8   273    5      1  250      0.5        0.5 1    1  1
```

First, for calculating empirical logit we'll only need the following variables: *trial*, *person*, *time*, *node*, and *yy*, so we'll create a new data frame caled *data.el* with these variables in this new order:

```
data.el <- data.node[,c(1,3,4,8,9)]
names(data.el) <- variable.names <- names(data.node[,c(1,3,4,8,9)])
head(data.el)
```

```
##   trial person time node yy
## 1   273      1  180    1  0
## 2   273      1  190    1  0
## 3   273      1  200    1  0
## 4   273      1  210    1  0
## 5   273      1  220    1  0
## 6   273      1  230    1  0
```

Line 1 above extracts the variables we need from *data.node*, which is all of them except for *item*, *contrast*, and *privileged*. Line 2 then sets the names of the variables in *data.el* to the names of the original variables in *data.node*.

Second, we'll create a matrix called *Empirical.Logit*, with each row having a unique combination of *trial*, *time*, and *node* (note that we don't include *person* here, because the empirical logit will be calculated across persons):

```
1  unique.trial <- sort(unique(data.el$trial))
2  unique.time <- sort(unique(data.el$time))
3  unique.node <- sort(unique(data.el$node))
4
5  number.variables <- 3
6
7  Empirical.Logit <- matrix(nrow=prod(length(unique.trial),length(unique.time),
8  length(unique.node)), ncol = (number.variables + 1))
```

Lines 1-3 above assign a variable to the unique values for *trial*, *time*, and *node* (respectively) in *data.el*. Line 5 specifies that the number of variables that we'll be calculating the empirical logit with is 3 (*trial*, *time*, and *node*). Line 7 creates the matrix *Empirical.Logit*, in which we'll insert every unique combination of *trial*, *time*, and *node*, as well as the empirical logit calculated for that combination. Notice that *Empirical.Logit* has a number of rows equal to the product of the number of unique values each variable possesses, and a number of columns equal to the number of variables plus one (the additional column will be used to input the calculated empirical logits).

Third, we'll calculate the empirical logits and input them into the *Empirical.Logit* matrix:

```
1  row.index <- 1
2
3  for (trial in 1:length(unique.trial)){
4  for (time in 1:length(unique.time)){
5  for (node in 1:length(unique.node)){
6
7  data.el.trial.time.node <- data.el[(data.el$trial==unique.trial[trial]
8  & data.el$time==unique.time[time] & data.el$node==unique.node[node]),]
9  yy.trial.time.node <- data.el.trial.time.node$yy
10 proportion <- sum(yy.trial.time.node[!is.na(yy.trial.time.node)])/length(yy.trial.time.node)
11 empirical.logit <- log(proportion/(1 - proportion))
12 Empirical.Logit[row.index,] <- c(unique.trial[trial], unique.time[time],
13 unique.node[node], empirical.logit)
14 row.index <- row.index + 1
15 }}}
16
17 Empirical.Logit <- data.frame(Empirical.Logit)
18 names(Empirical.Logit) <- c("trial", "time", "node", "empirical.logit")
19 Empirical.Logit$empirical.logit[Empirical.Logit$empirical.logit < -10^6] <- -10^6
20 Empirical.Logit$empirical.logit[Empirical.Logit$empirical.logit > 10^6] <- 10^6
```

Line 1 in the code above simply sets the initial value for the *row.index* we'll use to specify what row of *Empirical.Logit* we're currently using. Lines 3-5 indicate that we're repeating the following process for each unique combination of *trial*, *time*, and *node*. Line 7 extracts the rows of *data.el* that match the unique combination of *trial*, *node*, and *time* we're calculating the empirical logit for, calling this variable *data.el.trial.time.node*. Line 9 extracts the values for *yy* from *data.el.trial.time.node*, which will be used to calculate the empirical logit. Lines 10-11 calculate the empirical logit, using the formula $empirical.logit = log(\frac{proportion}{1-proportion})$, where the *proportion* is the proportion of *yy.trial.time.node* equal to one ($\frac{\sum yy}{length(yy)}$). Lines 17-18 turn *Empirical.Logit* into a dataframe, with the appropriate names for its variables. Lines 19-20 deal with any cases where $empirical.logit = \pm\infty$, which can occur when $proportion = 0$ or $proportion = 1$. Because the autocorrelations and partial autocorrelations cannot be calculated with infinite empirical logits, we set these values to extremely large values in magnitude (in this case, $\pm 10^6$).

Looking at *Empirical.Logit*, we should now have the calculated values for empirical logit:

```
##   trial time node empirical.logit
## 1     1  180    1       -1.791759
## 2     1  180    2       -2.564949
## 3     1  190    1       -1.791759
## 4     1  190    2       -2.564949
## 5     1  200    1       -1.791759
## 6     1  200    2       -2.564949
```

Calculation: Autocorrelation & Partial Autocorrelations

In this section we'll be calculating the autocorrelations and partial autocorrelations, using the empirical logits calculated above. Note that an autocorrelation will be calculated for each combination of *trial*, *node*, and *time.lag*. First, choose what you want your maximum time lag to be for the autocorrelations and partial autocorrelations. For this example, we'll use a maximum time lag of 20:

```
time.lag.max <- 20
```

Second, we'll set up the data frame *AC.PAC*, that will contain the autocorrelations (*AC*) and partial autocorrelations (*PAC*):

```
AC.PAC <- matrix(nrow=prod(length(unique.trial),length(unique.node),time.lag.max),
ncol=(number.variables + 2))
```

Notice that *AC.PAC* has a number of rows equal to the number of unique combinations of *trial*, *node*, and *time.lag* (ranging from 1 to *time.lag.max*), and a number of columns equal to the number of variables from before, plus two. The additional two columns are for inputting autocorrelations and partial autocorrelations.

Third, we'll calculate the autocorrelations and partial autocorrelations and input them into the *AC.PAC* matrix:

```
1  row.index <- 1
2
3  for (trial in 1:length(unique.trial)){
4  for (node in 1:length(unique.node)){
5
6  Empirical.Logit.trial.node <- Empirical.Logit[(Empirical.Logit$trial==unique.trial[trial]
7  & Empirical.Logit$node==unique.node[node]),]
8  autocorrelations <- acf(Empirical.Logit.trial.node$empirical.logit,
9  lag.max = time.lag.max, plot=FALSE)
10 autocorrelations <- autocorrelations$acf[2:(time.lag.max + 1)]
11 partial.autocorrelations <- pacf(Empirical.Logit.trial.node$empirical.logit,
12 lag.max = time.lag.max, na.action=na.pass, plot=FALSE)
13 partial.autocorrelations <- c(partial.autocorrelations$acf)
14 for (time.lag in 1:time.lag.max){
15 AC.PAC[(row.index + time.lag - 1),] <- c(unique.trial[trial], unique.node[node], time.lag,
16 autocorrelations[time.lag], partial.autocorrelations[time.lag])
17 }
18 row.index <- row.index + time.lag.max
19 }}
20
21 AC.PAC <- data.frame(AC.PAC)
22 names(AC.PAC) <- c("trial", "node", "time.lag", "autocorrelation", "partial.autocorrelation")
```

Line 1 above sets the initial value for *row.index*, used to reference which rows of *AC.PAC* we're calculating values for. Lines 3-4 indicate that we're repeating the following process for each unique combination of *trial* and *node*. Lines 6-7 extracts the rows of *Empirical.Logit* that match the unique combination of

trial and *node* we're calculating the autocorrelation and partial autocorrelation for, calling this variable *Empirical.Logit.trial.node*. Lines 8-14 calculate the autocorrelations and partial autocorrelations for each time lag, ranging from 1 to *time.lag.max*. Notice in line 10 that the autocorrelations are extracted for indices 2 through (*time.lag.max* + 1), this is because the *acf*() function includes results for *time.lag* = 0, which we aren't interested in for these purposes. Lines 14-17 include the autocorrelation and partial autocorrelations for each combination of *trial*, *node*, and *time.lag* into the *AC.PAC* matrix. Lines 21-22 turn *AC.PAC* into a dataframe, with the appropriate names for its variables.

Looking at *AC.PAC*, we should now have the calculated values for both autocorrelations and partial autocorrelations:

```
##   trial node time.lag autocorrelation partial.autocorrelation
## 1     1    1        1       0.9417591              0.94175909
## 2     1    1        2       0.8903962              0.03082523
## 3     1    1        3       0.8390333             -0.02378418
## 4     1    1        4       0.7876704             -0.02773465
## 5     1    1        5       0.7489045              0.08291772
## 6     1    1        6       0.7041814             -0.06349548
```

Plotting Figures: Autocorrelation & Partial Autocorrelations

In this step, we'll use ggplot2 to plot figures for the autocorrelations and partial autocorrelations calculated previously.

First, set the *time.lag* variable in *AC.PAC* as a factor:

```
AC.PAC$time.lag <- as.factor(AC.PAC$time.lag)
```

Second, for these plots, we will be plotting the results for an individual node. For this example, we'll plot the autocorrelations and partial autocorrelations for *node* = 1:

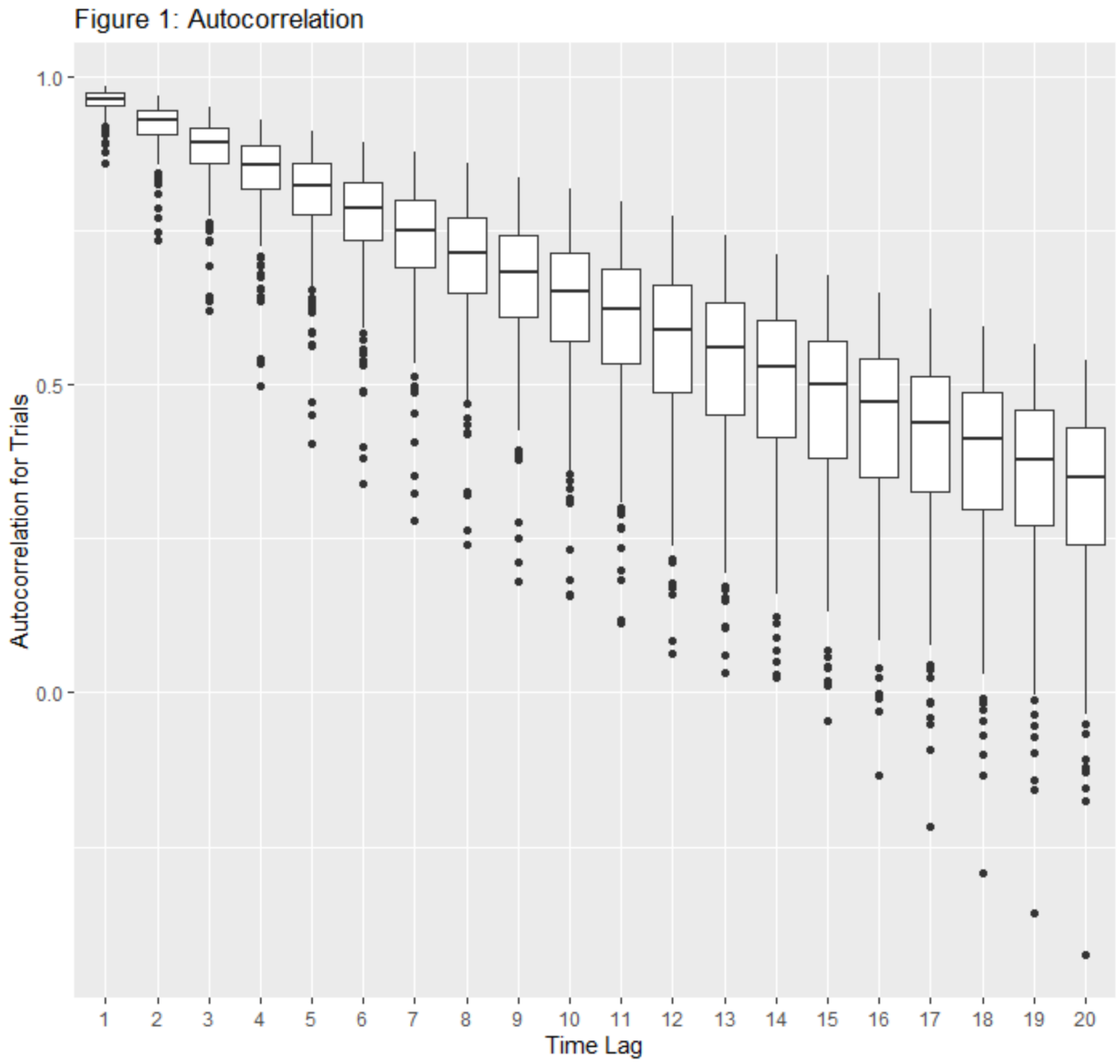
```
node.to.plot <- 1
AC.PAC.node <- AC.PAC[AC.PAC$node==node.to.plot,]
```

Change the value for *node.to.plot* above to the node you wish to plot. *AC.PAC.node* only gives the rows of *AC.PAC* with *node* = 1.

Third, plot the figures for autocorrelation and partial autocorrelation using ggplot2:

```
library(ggplot2)
AC.plot <- ggplot(AC.PAC.node, aes(x=time.lag, y=autocorrelation)) +
geom_boxplot() +
labs(title = "Figure 1: Autocorrelation", x = "Time Lag", y = "Autocorrelation for Trials")
```

Calling up *AC.plot* gives the following:

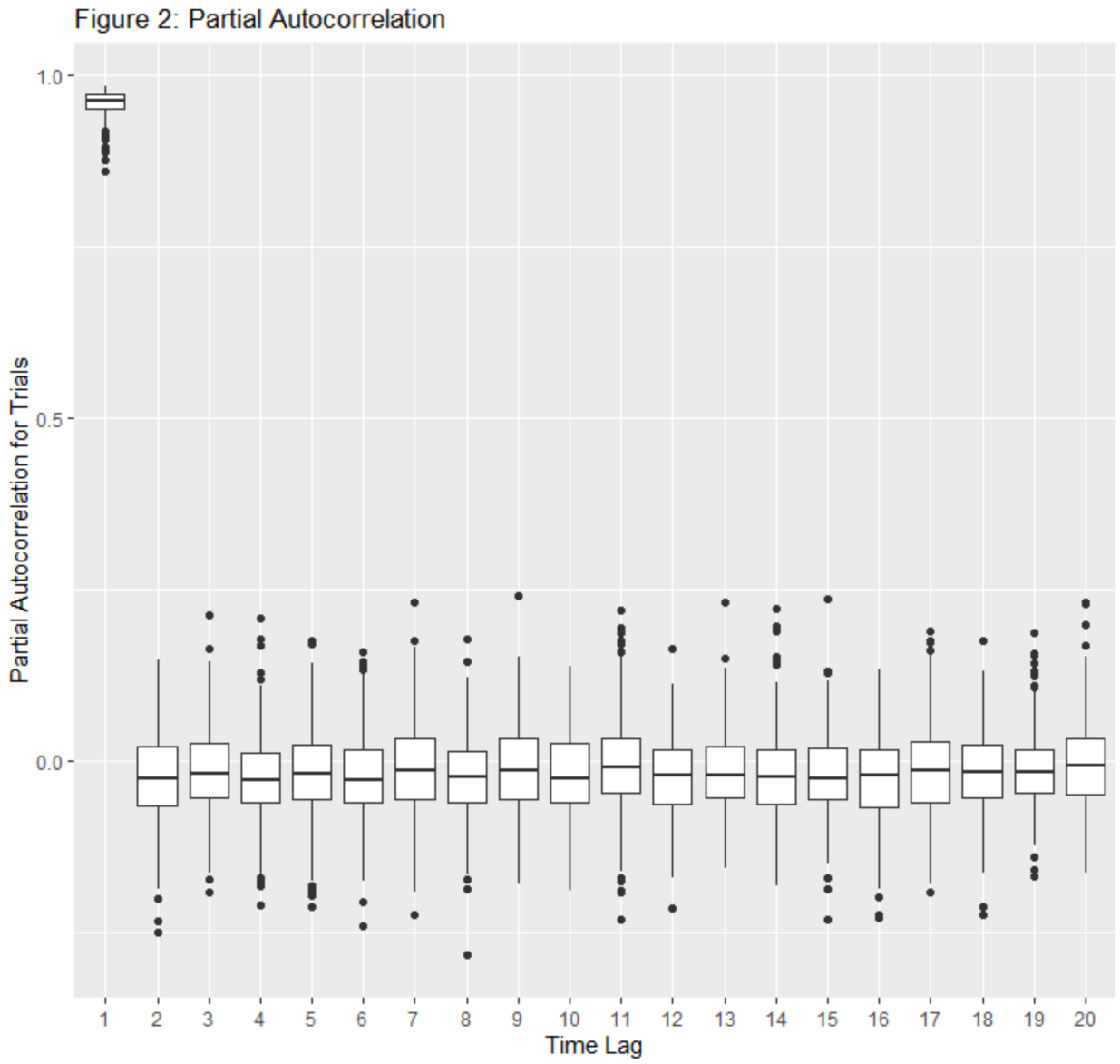


The box plot of correlograms across persons presented in Figure 1 above shows that the autocorrelations did not converge at zero, even at large values of lag. This pattern has also been observed when a time-series also exhibits trend (Chatfield, 2004, p. 26).

You should expect a figure somewhat similar to Figure 1 above, with the autocorrelation decreasing as a function of the time lag.

```
PAC.plot <- ggplot(AC.PAC.node, aes(x=time.lag, y=partial.autocorrelation)) +
geom_boxplot() +
labs(title = "Figure 2: Partial Autocorrelation", x = "Time Lag",
y = "Partial Autocorrelation for Trials")
```

Calling up *PAC.plot* gives the following:



You should expect a figure somewhat similar to Figure 2 above, with high partial autocorrelations at $time.lag = 1$ that drop off quickly for $time.lag > 1$. This suggests that the first order of autocorrelations should be modeled.

Plotting Figures: Trend

In this section, we'll use ggplot2 to plot figures for the values of empirical logit we calculated above. Each figure will illustrate the empirical logit across trials (for a particular node) over time.

First, set the *time* variable in *Empirical.Logit* as a factor:

```
Empirical.Logit$time <- as.factor(Empirical.Logit$time)
```

Second, for this plots, you will be plotting the results for an individual node. For this example, we'll plot the empirical logits for $node = 1$:

```
node.to.plot <- 1
Empirical.Logit.node <- Empirical.Logit[Empirical.Logit$node==node.to.plot,]
```

Change the value for *node.to.plot* above to the node you wish to plot. *Empirical.Logit.node* only gives the rows of *Empirical.Logit* with $node = 1$.

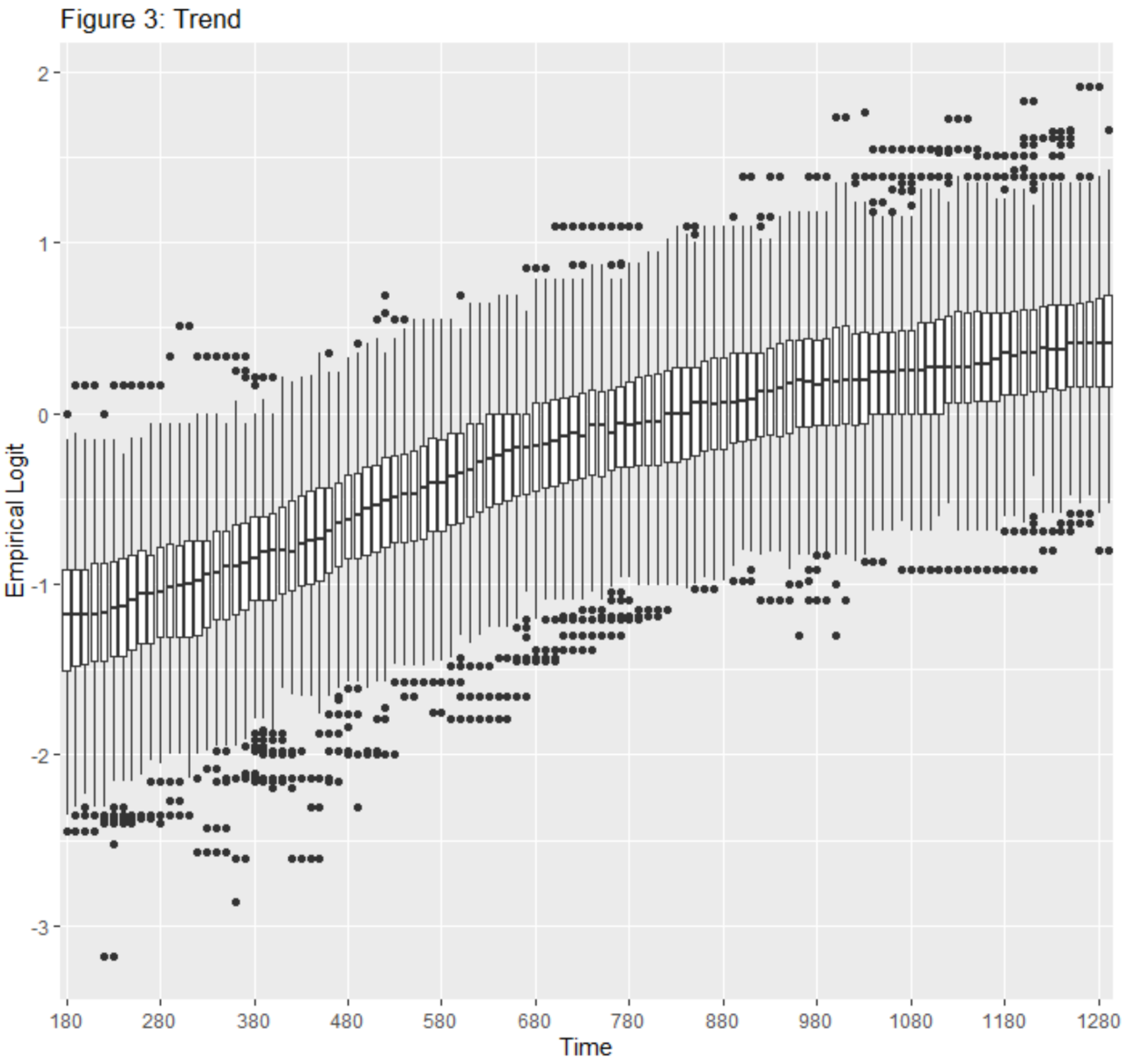
Third, choose what breaks in the x-axis you want for your figure. This step is optional, but if you have more than ~20 unique time points, this will keep the x-axis's labels from becoming cluttered. In this example, *time* ranges from 180 to 1290 in increments of 10 (for a total of 112 time points). To reduce this number for the figure I only use time points in increments of 100 (by setting $by = 100$):

```
time.breaks <- seq(min(as.numeric(levels(Empirical.Logit$time))),
                   max(as.numeric(levels(Empirical.Logit$time))),by=100)
```

Fourth, plot the figure for empirical logit using ggplot2:

```
library(ggplot2)
Empirical.Logit.plot <- ggplot(Empirical.Logit.node, aes(x=time, y=empirical.logit)) +
geom_boxplot() +
labs(title = "Figure 3: Trend", x = "Time", y = "Empirical Logit") +
scale_x_discrete(breaks=time.breaks)
```

Calling up *Empirical.Logit.plot* gives the following:



You should expect a figure somewhat resembling Figure 3 above. The overall trend was positive and (approximatley) linear over time. Fitted lines over time were similar between the linear function and Kernal-weighted local polynomial smoothing function, and only small deviations from the linear trend were observed.

Similar patterns in autocorrelations, partial autocorrelations, and trend were observed for $node = 2$. Based on exploratory data analyses of change processes (linear trend and the first-order autocorrelations), we consider a dynamic IRTree model which accounts for the change processes.

Model Specification and Estimation

In this section, we will first set up our data to include covariate coding for trend and autocorrelations. We will then specify dynamic IRTree models with these autocorrelation effects using the glmer function in the lme4 package (Bates, Mächler, Bolker, & Walker, 2015).

First, using *data.node* that we created in the Data Management step:

```
##   trial item person time contrast privileged y node yy
## 1   273    5      1  180      0.5         0.5 3    1  0
## 2   273    5      1  190      0.5         0.5 3    1  0
## 3   273    5      1  200      0.5         0.5 3    1  0
## 4   273    5      1  210      0.5         0.5 3    1  0
## 5   273    5      1  220      0.5         0.5 3    1  0
## 6   273    5      1  230      0.5         0.5 3    1  0
```

We are going to define and create the following 9 new variables:

- *node*.1 - Dummy coded variable for *node*, where *node*.1 = 1 if *node* = 1 and *node*.1 = 0 if *node* = 2.
- *node*.2 - Dummy coded variable for *node*, where *node*.2 = 0 if *node* = 1 and *node*.2 = 1 if *node* = 2.
- t - Whether the person is looking at the target. $t = 1$ if $y = 1$, and $t = 0$ otherwise.
- c - Whether the person is looking at the competitor. $c = 1$ if $y = 2$, and $c = 0$ otherwise.
- *time.coded* - The ordinal positionings of *time* (e.g., $0, 1, 2, 3, \ldots$) for trend effect.
- *t.lag* - The value of t at the previous time point. Undefined for the first time point.
- *c.lag* - The value of c at the previous time point. Undefined for the first time point.
- *yy.lag* - The value of *yy* at the previous time point. Undefined for the first time point.
- *time.coded.centered* - The values of *time.coded* centered within a cluster (a unique combination of *trial* and *person*) at each *node* (e.g., *time.coded* values of $1, 2, 3, \ldots, 98, 99, 100$ becomes *time.coded.centered* values of $-49.5, -48.5, -47.5, \ldots, 47.5, 48.5, 49.5$).

Notice that for the lag variables listed above (*yy.lag*, *t.lag*, and *c.lag*) we will be using deviation coding, meaning that instead of being coded as 0 or 1 (to reflect the previous values of their respective variables), they will be coded as -1 or 1. For example, if $yy = 0$ at *time.coded* = 3, then *yy.lag* $= -1$ at *time.coded* = 4, or if $yy = 1$ at *time.coded* = 3, then *yy.lag* = 1 at *time.coded* = 4. The 9 variables listed above are illustrated in the following table for the first 10 time points for a single combination of *trial* and *person*:

y	time	*time.coded*	*time.coded.centered*	*yy*	*t*	*c*	*yy.lag*	*t.lag*	*c.lag*
1	180	0	-54.5	1	1	0	-	-	-
2	190	1	-53.5	1	0	1	1	1	-1
1	200	2	-52.5	1	1	0	1	-1	1
3	210	3	-51.5	0	0	0	1	1	-1
3	220	4	-50.5	0	0	0	-1	-1	-1
2	230	5	-49.5	1	0	1	-1	-1	-1
2	240	6	-48.5	1	0	1	1	-1	1
3	250	7	-47.5	0	0	0	1	-1	1
1	260	8	-46.5	1	1	0	-1	-1	-1
2	270	9	-45.5	1	0	1	1	1	-1

y	time	*time.coded*	*time.coded.centered*	*yy*	*t*	*c*	*yy.lag*	*t.lag*	*c.lag*
1	180	0	-49.0	1	1	0	-	-	-
2	190	1	-48.0	0	0	1	1	1	-1
1	200	2	-47.0	1	1	0	-1	-1	1
3	210	3	-	NA	-	-	NA	-	-
3	220	4	-	NA	-	-	NA	-	-
2	230	5	-46.0	0	0	1	1	-1	-1
2	240	6	-45.0	0	0	1	-1	-1	1
3	250	5	-	NA	-	-	NA	-	-
1	260	8	-44.0	1	1	0	-1	-1	-1
2	270	9	-43.0	0	0	1	1	1	-1

The top and bottom sections of the table above are for *node* = 1 and *node* = 2, respectively. Notice that observations for $y = 3$ and *node* = 2 are undefined, and are omitted for the coding of *time.coded.centered* and lag effects. We'll explain this later on in this section when we remove these observations from the data.

First, we're going to reorder *data.node* by *node*, *person*, *trial*, and *time*. This will make the following code easier to implement:

```
data.node <- data.node[order(data.node$node, data.node$person, data.node$trial, data.node$time),]
```

Second, we're going to use zeroes as placeholders to add our 9 variables to *data.node*:

```
data.node$node.1 <- data.node$node.2 <- data.node$yy <- data.node$t <-
data.node$c <- data.node$time.coded <- data.node$t.lag <- data.node$c.lag <-
data.node$yy.lag <- data.node$time.coded.centered <- rep(0,nrow(data.node))
```

Third, we're going to add the *node*.1, *node*.2, t, and c variables:

```
t.list <- list(1,0,0)
c.list <- list(0,1,0)

data.node$node.1[which(data.node$node==1)] <- 1
data.node$node.2[which(data.node$node==2)] <- 1

for (y.value.index in 1:length(y.values)){
y.value <- y.values[y.value.index]
match <- which(data.node$y==y.value)
data.node$t[match] <- t.list[[y.value]]
data.node$c[match] <- c.list[[y.value]]
}}
```

Lines 1-2 above set up the values of t and c for different values of y. Lines 4-5 create the *node*.1 and *node*.2 variables in our data. Lines 7-12 create the t and c variables for each value of y, using *t.list* and *c.list*. For example, when the target is observed ($y = 1$), $t = t.list[[1]] = 1$ and $c = c.list[[1]] = 0$.

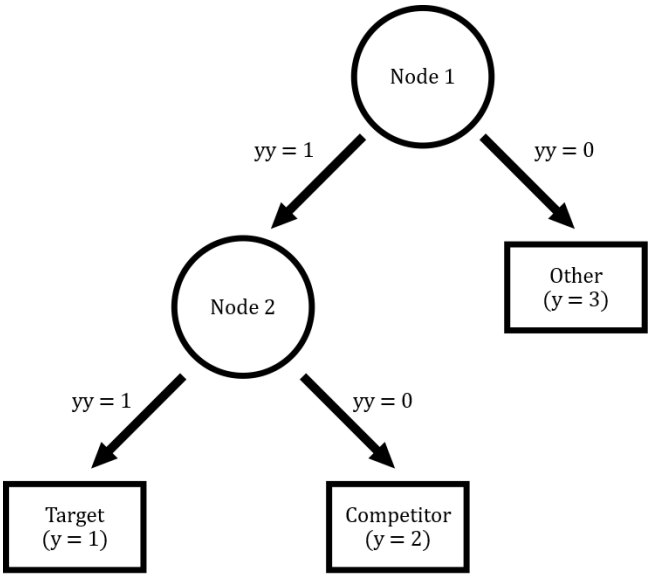
Fourth, we're going to add the *t.lag*, *c.lag*, and *time.coded* variables:

```
unique.person <- sort(unique(data.node$person))
unique.trial <- sort(unique(data.node$trial))
deviation.coding <- c(-1,1)

for (node in 1:number.nodes){
for (person in 1:length(unique.person)){
for (trial in 1:length(unique.trial)){
match <- which(data.node$node==node & data.node$person==unique.person[person]
& data.node$trial==unique.trial[trial])
if (length(match)>0){
time.data <- data.node$time[match]
t.data <- data.node$t[match]
c.data <- data.node$c[match]
data.node$t.lag[match] <- c(NA, deviation.coding[t.data[1:(length(t.data)-1)] + 1])
data.node$c.lag[match] <- c(NA, deviation.coding[c.data[1:(length(c.data)-1)] + 1])
time.coded.matrix <- matrix(c(seq(0,(length(time.data)-1)),time.data),ncol=2)
time.coded <- time.coded.matrix[order(time.coded.matrix[,2]),1]
data.node$time.coded[match] <- time.coded
}}}}
```

In lines 1-2 above, we again use the unique values of *person* and *trial*, because the values for *t.lag*, *c.lag*, and *time.coded* are defined for the values of *time* for a single person on a single trial (varying by node). Line 3 above sets up the deviation coding we'll use for lag effects. Lines 5-7 indicate that the loop will repeat for each unique combination of *node*, *person*, and *trial*. Line 8 extracts the observations from *data.node* matching the values for these three variables. The 'if' statement in line 10 is included because not every person completed every trial, so we only run the loop on combinations of *person* and *trial* with at least one observation. Lines 11-13 extract the values of *time*, t, and c we'll need to create the following variables. Lines 14-15 create the *t.lag* and *c.lag* deviation-coded variables. Notice that the lag effects are undefined at the initial time point. Lines 16-18 create the *time.coded* variable, starting at 0 for the initial time point (this time point will later be removed).

Fifth, we're going to remove the observations from *data.node* with $node = 2$ and $y = 3$. As shown in the figure below, these observations aren't defined for this combination of variables and should be removed from the data.




```
if (length(which(data.node$node==2 & data.node$y==3))>0){
data.node <- data.node[-which(data.node$node==2 & data.node$y==3),]
}
```

Sixth, we're going to create the *yy.lag* variable (*after* removing the observations above), much in the same way we created *t.lag* and *c.lag*:

```
for (node in 1:number.nodes){
for (person in 1:length(unique.person)){
for (trial in 1:length(unique.trial)){
match <- which(data.node$node==node & data.node$person==unique.person[person]
& data.node$trial==unique.trial[trial])
if (length(match)>0){
yy.data <- data.node$yy[match]
data.node$yy.lag[match] <- c(NA, deviation.coding[yy.data[1:(length(match)-1)] + 1])
}}}}
```

Seventh, we're going to remove the initial values for time (of which the lag variables are undefined) for each combination of *node*, *person*, and *trial*, after adding our *yy.lag* variable:

```
for (node in 1:number.nodes){
for (person in 1:length(unique.person)){
for (trial in 1:length(unique.trial)){
match <- which(data.node$node==node & data.node$person==unique.person[person]
& data.node$trial==unique.trial[trial])
if (length(match)>0){
time.data <- data.node$time[match]
start.time <- min(time.data)
start.time.index <- match[which(time.data==start.time)]
data.node <- data.node[-start.time.index,]
}}}}
```

Line 8 above determines the smallest value of *time* (e.g., the initial time point) for a person's performance on

a trial for a given node. Line 9 then removes this initial observation from *data.node*.

Eighth, we're going to add the *time.coded.centered* variable. Note that we add this after the initial time point has been removed.

```
for (node in 1:number.nodes){
for (person in 1:length(unique.person)){
for (trial in 1:length(unique.trial)){
match <- which(data.node$node==node & data.node$person==unique.person[person]
& data.node$trial==unique.trial[trial])
if (length(match)>0){
time.coded.data <- data.node$time.coded[match]
data.node$time.coded.centered[match] <- time.coded.data - mean(time.coded.data)
}}}}
```

Line 8 takes the values of *time.coded* within a cluster (for a unique combination of *trial* and *person* at each node), and subtracts the mean from those values to center them about 0.

Now that we've created all of the variables needed to run the models, we'll set the appropriate variables be treated as factors or numeric data so that they will be used by glmer correctly:

```
data.node$yy.lag <- as.numeric(data.node$yy.lag)
data.node$t.lag <- as.numeric(data.node$t.lag)
data.node$c.lag <- as.numeric(data.node$c.lag)
data.node$time.coded <- as.numeric(data.node$time.coded)
data.node$time.coded.centered <- as.numeric(data.node$time.coded.centered)
data.node$item <- as.factor(data.node$item)
data.node$person <- as.factor(data.node$person)
data.node$trial <- as.factor(data.node$trial)
data.node$node <- as.factor(data.node$node)
data.node$node.1 <- as.factor(data.node$node.1)
data.node$node.2 <- as.factor(data.node$node.2)
```

Notice above that the experimental factors in *data.node* are coded numerically, and the covariates are coded as factors.

The variables are now prepared to fit the models. In the original data, there are two covariates regarding experimental conditions: *contrast* and *privileged*. These variables are Helmert-coded contrasts to distinguish three different conditions used. The coding of these variables for the three conditions is presented below:

Condition:	*contrast*	*privileged*
One-Contrast	-1	0
Two-Contrasts Privileged	0.5	0.5
Two-Contrasts Shared	0.5	-0.5

The first model we'll run (Model.1) uses the *t.lag* and *c.lag* variables:

```
library(lme4)

Model.1 <- glmer(yy ~ -1 + node + t.lag:node + c.lag:node + privileged:node
+ contrast:node + time.coded:node + (-1 + node |trial) +
(-1+t.lag:node + c.lag:node+node|person) + (-1 + t.lag:node + c.lag:node + node|item),
family = binomial,
data = data.node)

summary(Model.1)
```

The syntax for Model.1 is based on the following equation:

$$\begin{aligned}
&\text{logit}[P(y^{*}_{tljir}|x_{T(t-1)ljir}, x_{C(t-1)ljir}, time_{tljir}, \mathbf{x}, \delta_{ljir}, \lambda_{1jr}, \theta_{jr}, \lambda_{2ir}, \beta_{ir})] \\
&= [x^{'}_{T(t-1)ljir}\lambda_{Tr} + x^{'}_{C(t-1)ljir}\lambda_{Cr} + time^{'}_{tljir}\zeta_r + \mathbf{x}^{'}\gamma_r] \\
&+ \delta_{ljir} + [x^{'}_{T(t-1)ljir}\lambda_{T1jr} + x^{'}_{C(t-1)ljir}\lambda_{C1jr} + \theta_{jr}] + [x^{'}_{T(t-1)ljir}\lambda_{T2ir} + x^{'}_{C(t-1)ljir}\lambda_{C2ir} + \beta_{ir}], \qquad (1)
\end{aligned}$$

where t is an index for time, l is an index for trial, j is an index for person, i is an index for item, and r is an index for node.

In the syntax for Model.1, the binary variable yy is predicted using the following fixed and random effects:

- Fixed effects
  - node for the intercept γ_r
  - t.lag:node for the fixed lag effect λ_{Tr}
  - c.lag:node for the fixed lag effect λ_{Cr}
  - privileged:node for the first experimental condition γ_{1r}
  - contrast:node for the second experimental condition γ_{2r}
  - time.coded:node for the fixed trend effect ζ_r
- Random effects
  - (-1 + node | trial) for the trial random effect δ_{ljir}
  - (-1 + t.lag:node + c.lag:node + node | person) for random person effects λ_{Tr}, λ_{Cr}, and θ_{jr}
  - (-1 + t.lag:node + c.lag:node + node | item) for random item effects λ_{Tr}, λ_{Cr}, and β_{ir}

The argument *family = binomial* in the model syntax specifies the random component for binomial data.

The second model we'll run (Model.2) uses the *yy.lag* variables:

```
library(lme4)

Model.2 <- glmer(yy ~ -1 + node + yy.lag:node +  privileged:node
+ contrast:node + time.coded:node +(-1+node|trial) +
(-1+yy.lag:node+node|person) + (-1 + yy.lag:node + node|item), family = binomial,
data = data.node)

summary(Model.2)
```

The syntax for Model.2 is based on the following equation:

$$\begin{aligned}
&logit[P(y^{*}_{tljir}|y^{*}_{(t-1)ljir}, time_{tljir}, x, \delta_{ljir}, \lambda_{1jr}, \theta_{jr}, \lambda_{2ir}, \beta_{ir})] \\
&= [y^{*'}_{(t-1)ljir}\lambda_r + time^{'}_{tljir}\zeta_r + x^{'}\gamma_r] + \delta_{ljir} + [y^{*'}_{(t-1)ljir}\lambda_{1jr} + \theta_{jr}] + [y^{*'}_{(t-1)ljir}\lambda_{2ir} + \beta_{ir}] \qquad (2)
\end{aligned}$$

In the syntax for Model.2, the binary variable yy is predicted using the following fixed and random effects:

- Fixed effects
  - node for the intercept γ_r
  - yy.lag:node for the fixed lag effect λ_r
  - privileged:node for the first experimental condition γ_{1r}
  - contrast:node for the second experimental condition γ_{2r}
  - time.coded:node for the fixed trend effect ζ_r
- Random effects
  - (-1 + node | trial) for the trial random effect δ_{ljir}
  - (-1 + yy.lag:node + node | person) for random person effects λ_r and θ_{jr}
  - (-1 + yy.lag:node + node | item) for random item effects λ_r and β_{ir}

		Trial	Person		Item	
Model	Node Specific?	Trial Int.	Person Int.	Slope	Item Int.	Slope
Model A	No	√	√		√	
Model B	Yes	√	√		√	
Model B*	Yes	√	√		√	
Model B* Person	Yes	√	√	√	√	
Model B* Item	Yes	√	√		√	√
Model B* Person & Item	Yes	√	√	√	√	√

Model Selection

In this section, we'll fit 6 models using glmer, first with the *t.lag* and *c.lag* variables (similar to Model 1 in the previous section), and again with the *y.lag* variable (similar to Model 2 in the previous section). The differences in the effects included in each of these 6 models is summarized in the above table.

All 6 models include random intercepts for *trial* (`(1|trial)`), *person* (`(1|person)`), and *item* (`(1|item)`). Differences among the models include which (if any) node-related variables (*node* or *node*.2) are used, and whether random person or item lag slopes are used, as summarized in the table above.

First, we'll run these 6 models using the *t.lag* and *c.lag* variables.

```
Model.A.1 <- glmer(yy ~ 1 + t.lag + c.lag + time.coded.centered + (1|trial) + (1|person)
+ (1|item), family = binomial, data = data.node)
```

Model.A.1 is the simplest model we'll run. It ignores values of *node*, and does not use random person or item lag slopes.

```
Model.B.1 <- glmer(yy ~ -1 + node + t.lag:node + c.lag:node + time.coded.centered:node
+ (-1+node|trial) + (-1+node|person) + (-1+node|item), family = binomial, data = data.node)
```

Model.B.1 includes the values of *node*, allowing values of *t.lag*, *c.lag* (`t.lag:node` and `c.lag:node`), trial, person, and item intercepts [`(-1+node|trial)`, `(-1+node|item)`, and `(-1+node|person)`], and *time.coded.centered* (`time.coded.centered:node`) to vary by node.

```
Model.B.star.1 <- glmer(yy ~ -1 + node + t.lag:node + c.lag:node + time.coded.centered:node
+ (-1+node.2|trial) + (-1+node|person) + (-1+node|item), family = binomial, data = data.node)
```

Model.B.star.1 is identical to *Model.B*.1, but now uses the dummy-coded variable *node*.2 for the trial intercept `(-1+node.2|trial)`, because the estimate of the variance of the random trial intercept for *node* = 1 was near the boundary. The following three models are extensions of *Model.B.star*.1, using random person or item lag slopes.

```
Model.B.star.Person.1 <- glmer(yy ~ -1 + node + t.lag:node + c.lag:node
+ time.coded.centered:node + (-1+node.2|trial) + (-1+t.lag:node+c.lag:node+node|person)
+ (-1+node|item), family = binomial, data = data.node)
```

Model.B.star.Person.1 is an extension of *Model.B.star*.1 including the random person lag slope `(-1+t.lag:node+c.lag:node+node|person)`.

```
Model.B.star.Item.1 <- glmer(yy ~ -1 + node + t.lag:node + c.lag:node
+ time.coded.centered:node + (-1+node.2|trial) + (-1+node|person)
+ (-1+t.lag:node+c.lag:node+node|item), family = binomial, data = data.node)
```

Model.B.star.Item.1 is an extension of *Model.B.star*.1 including the random item lag slope `(-1+t.lag:node+c.lag:node+node|item)`.

```
Model.B.star.Person.Item.1 <- glmer(yy ~ -1 + node + t.lag:node + c.lag:node
+ time.coded.centered:node + (-1+node.2|trial) + (-1+t.lag:node+c.lag:node+node|person) +
(-1+t.lag:node+c.lag:node+node|item), family = binomial, data = data.node)
```

Model.B.star.Person.Item.1 is an extension of *Model.B.star*.1 including both the random person and item lag slopes [`(-1+t.lag:node+c.lag:node+node|person)` and $`(-1+t.lag:node+c.lag:node+node|item)`]$.

We can use the anova function to compare the fit of these models:

```
anova(Model.A.1, Model.B.1, Model.B.star.1, Model.B.star.Person.1,
Model.B.star.Item.1, Model.B.star.Person.Item.1)
```

Using the anova function, AIC and BIC for these models can be obtained.

Second, we'll run these 6 models using the *yy.lag* variable. These models are identical to the first 6 discussed in regards to what variables and effects are used, except with the substitution of *c.lag* and *t.lag* for *yy.lag*.

```
Model.A.2 <- glmer(yy ~ 1 + yy.lag + time.coded.centered + (1|trial) + (1|person) +
(1|item), family = binomial, data = data.node)
```

```
Model.B.2 <- glmer(yy ~ -1 + node + yy.lag:node + time.coded.centered:node
+ (-1+node|trial) + (-1+node|person) + (-1+node|item), family = binomial, data = data.node)
```

```
Model.B.star.2 <- glmer(yy ~ -1 + node + yy.lag:node + time.coded.centered:node +
(-1+node.2|trial) + (-1+node|person) + (-1+node|item), family = binomial,
data = data.node)
```

```
Model.B.star.Person.2 <- glmer(yy ~ -1 + node + yy.lag:node + time.coded.centered:node +
(-1+node.2|trial) + (-1+yy.lag:node+node|person) +
(-1+node|item), family = binomial,
data = data.node)
```

```
Model.B.star.Item.2 <- glmer(yy ~ -1 + node + yy.lag:node + time.coded.centered:node +
(-1+node.2|trial) + (-1+node|person) + (-1+yy.lag:node+node|item),
family = binomial, data = data.node)
```

```
Model.B.star.Person.Item.2 <- glmer(yy ~ -1 + node + yy.lag:node +
time.coded.centered:node + (-1+node.2|trial) + (-1+yy.lag:node+node|person) +
(-1+yy.lag:node+node|item), family = binomial, data = data.node)
```

Again, we can use the anova function to compare the fit of these models:

```
anova(Model.A.2, Model.B.2, Model.B.star.2, Model.B.star.Person.2,
Model.B.star.Item.2, Model.B.star.Person.Item.2)
```

Using the anova function, AIC and BIC for these models can be obtained.

Model Evaluation

Based on the model comparisons made in the previous section, we selected *Model*.2*b* as our final model. This model is identical to *Model*.2 in the Model Specification and Estimation section, but with `(-1 + node.2 | trial)` substituted for `(-1 + node | trial)`, because the estimate of the variance of the random trial intercept for *node* = 1 was near the boundary. The syntax for *Model*.2*b* is as follows:

```
Model.2b <- glmer(yy ~ -1 + node + yy.lag:node +  privileged:node
+ contrast:node + time.coded:node +(-1+node.2|trial) +
(-1+yy.lag:node+node|person) + (-1 + yy.lag:node + node|item), family = binomial,
data = data.node)
```

Descriptions of the fixed and random effects of *Model*.2*b* are identical to those provided for *Model*.2 in the Model Specification and Estimation section, again with the substituion of `(-1 + node.2 | trial)` for `(-1 + node | trial)`.

In this section, we will evaluate $Model.2b$ to determine whether this model accurately describes the data by calculating the standardized residuals of $Model.2b$, calculating the Somers' rank correlation, and by plotting figures comparing the data and predicted values (on the logit scale) from $Model.2b$.

Residual Analysis

In this section we'll calculate the standardized residuals obtained by $Model.2b$. First, we convert the fitted values of $Model.2b$ from the logit scale to the probability scale (denoted by $probs$):

```
probs <- 1/(1 + exp(-fitted(Model.2b)))
```

Second, using the $probs$ variable we'll calculate the standardized residuals using the following equation:

$$Standardized\ Residuals = \frac{yy - probs}{\sqrt{probs \cdot (1 - probs)}} \tag{3}$$

Note that to standardize residuals we divide the residuals $(yy - probs)$ by the standard deviation, which (for binomial data) is $\sqrt{probs \cdot (1 - probs)}$.

```
std.residuals <- (yy - probs)/sqrt(probs*(1-probs))
```

Any standardized residuals with an absolute value larger than 1.96 are indicative of potential misfit at the 5% level. The following code determines if any of these standardized residuals are too large (and if so, which ones):

```
length(which(abs(std.residuals)>1.96))
which(abs(std.residuals)>1.96)
```

If there is no misfit, then the above code will return values of 0.

Somers' Rank Correlation

The Somers' rank correlation is a measure of the ordinal predictive power of $Model.2b$. The Somers' rank correlation between the estimated probabilities ($probs$) and the data yy (which is a binary variable) can be calculated using the somers2 function in the Hmisc package (Harrell, 2020):

```
library(Hmisc)
somers2(probs, as.numeric(na.omit(data.node$yy)))
```

Note that the outputted value Dxy is the Somers' rank correlation.

Data vs. Predicted Values Figure

In this section we will plot figures comparing the predicted values from $Model.2b$ with the data yy at varying time points. These figures can be generated for either node for an individual person (with the predicted values calculated across items) or for an individual item (with the predicted values calculated across persons).

As an example we'll plot the proportions (i.e., a summary of the data) for $person = 1$ at both nodes. First, we'll construct the data frame containing these proportions for $person = 1$ and $node = 1$:

```
1  model.fitted <- unlist(unname(fitted(Model.2b)))
2  YY <- data.node$yy
3
4  person.1 <- which(data.node$person == 1 & data.node$node==1)
5  data.node.person.1 <- data.node[person.1,]
6  proportions.person.1 <- model.fitted[person.1]
7  YY.person.1 <- YY[person.1]
8  time.coded.person.1 <- sort(unique(data.node.person.1$time.coded))
9
10 proportions.data <- predicted <- c()
11 for (time.point in 1:length(time.coded.person.1)){
12 proportions.data[time.point] <- mean(YY.person.1[which(data.node.person.1$time.coded==
13 time.coded.person.1[time.point])])
14 predicted[time.point] <- mean(proportions.person.1[which(data.node.person.1$time.coded==
15 time.coded.person.1[time.point])])
16 }
17
18 model.fit.table <- data.frame(rep(time.coded.person.1, 2), c(proportions.data, predicted),
19 c(rep("Data",length(Time)),rep("Predicted Values",length(Time))))
20 names(model.fit.table) <- c("Time", "Proportion", "Group")
```

Lines 1-2 above set the values for the predicted values calculated from the model ($model.fitted$) and the values of yy from the data (YY) as variables. Lines 4-8 extract the values for $model.fitted$, YY, and $time.coded$ specifically for data points where $person = 1$ and $node = 1$. Lines 10-16 calculate the proportion at each time point as the average of person 1's responses for all items ($proportions.data[time.point]$), and as the average of the proportions predicted by $Model.2b$ at each time point ($predicted[time.point]$). Lines 18-20 assemble these proportions into a data frame usable by ggplot2.

The code above can be adapted for $node = 2$ (by changing $data.node\$node == 1$ to $data.node\$node == 2$ in line 4 above), for different persons (by changing $data.node\$person == 1$ in line 4 above), or for items instead of persons (by changing $data.node\$person == 1$ to $data.node\$item == 1$ in line 4 above).

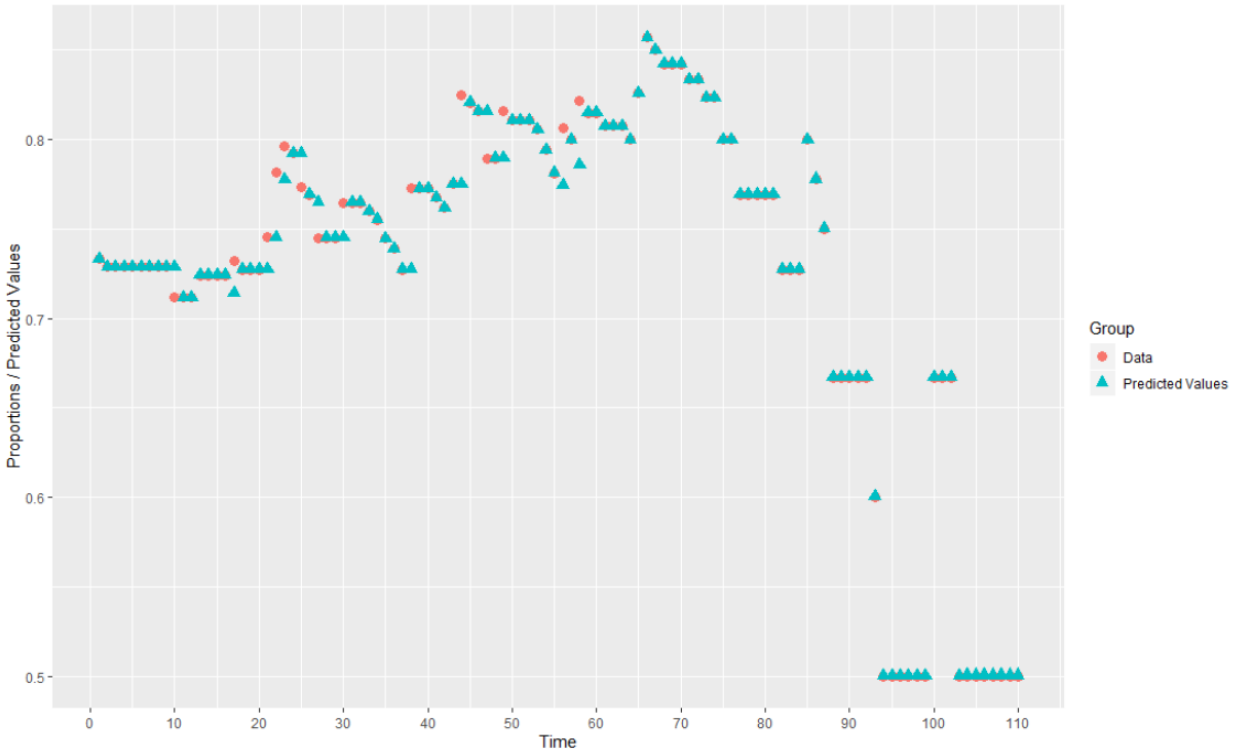
Second, we'll use our data frame ($model.fit.table$) to plot the figure using ggplot2:

```
library(ggplot2)
model.fit.plot <- ggplot(data = model.fit.table, aes(x = Time, y = Proportion,
group = Group)) +
  geom_point(aes(shape=Group,color=Group),size=2) +
  scale_x_continuous(breaks=seq(0,max(Time),by=10)) +
  scale_y_continuous(breaks=seq(0,1,by=.1))
```

Calling up $model.fit.plot$ should give a figure similar to the following:

If the model accurately predicts the probabilities for $person = 1$ at $node = 1$, then the plots for the data and the predicted values (the red and blue points respectively) should be close at each time point.

Repeating the above process for $person = 1$ at $node = 2$ results in the following figure:



References

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.

Chatfield, C. (2004). *The analysis of time series : An introduction.* (6th ed.). Boca Raton, FL: Chapman & Hall/CRC.

Cho, S.-J., Brown-Schmidt, S., De Boeck, P., & Shen, J. (2020). Modeling intensive polytomous time series eye-tracking data: A dynamic tree-based item response model. *Psychometrika, 85,* 154-184.

Harrell, F. (2020). Hmisc: Harrell miscellaneous. https://CRAN.R-project.org/package=Hmisc.

Wickham, H. (2016). Ggplot2: Elegant graphics for data analysis. New York, NY: Springer-Verlag. https://ggplot2.tidyverse.org.