Ch 7: Vector Autoregressive Models with Time-Varying AR Coefficients in dynr

Sy-Miin Chow

June 5, 2021

Contents


```
#Start fresh, set display options and load required libraries
rm(list=ls())
set.seed(12345678)
options(scipen = 999, digits=5)
library(mvtnorm)
library(dynr)
library(mgcv)
library(quantmod)
```

Background

Introduction

This example uses simulated data inspired by the empirical illustration in Chen, Chow, Hammal, Cohn, & Messinger (2021). In their example, a vector autoregressive model (VAR(1)) of order 1 with time-varying autoregressive (AR) coefficients was used to represent the interactive dynamics between mother and infant during the face-to-face/still-face (FFSF) protocol. Baby's and mother's AR coefficients are modeled as

varying within person over time and across individuals based on the operating FFSF episode at the moment, whether the mother was detected to be smiling, and the infant's level of attachment. Here, we create a simpler simulation example with one time-covariate, x_{it}, that drives the changes in the dyad members' AR coefficients.

For first-time users of dynr

The dynr package compiles C code in response to user input, so more setup is required for the dynr package than for many others, including installation of the GSL library and C compiler. Steps for installation and automated installer for Windows can be downloaded here: https://dynrr.github.io/predownload.html

Reference

Chen, M., Chow, S-M., Hammal, Z., Cohn, J. & Messinger, D. (2021). A Person- and time-varying vector autoregressive model to capture interactive infant-mother head movement dynamics. Multivariate Behavioral Research. DOI:10.1080/00273171.2020.1762065

Chow, S-M. *Zu, J., Shifren, K. & Zhang, G. (2011). Dynamic factor analysis models with time-varying parameters. Multivariate Behavioral Research, 46(2), 303-339.

Gates, K., Chow, S-M., & Molenaar, P.C.M. Analysis of Intraindividual Variation. Systems Approaches to Human Process Analysis. New York, NY: Taylor & Francis.

Molenaar PC, Sinclair KO, Rovine MJ, Ram N, Corneal SE. Analyzing developmental processes on an individual level using nonstationary time series modeling. Developmental Psychology. 45: 260-71. PMID 19210007 DOI: 10.1037/A0014170

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. https://journal.r-project.org/archive/2019/RJ-2019-012/index.html

Dynr development team. Dynamic modeling in R (dynr) website. https://dynrr.github.io/

Dynr development team. User Installation Instructions Manual for dynr.

https://cran.r-project.org/web/packages/dynr/vignettes/InstallationForUsers.pdf

Simulated Data Generation

Initial set-up

Here we define the number of time points, participants, true parameter values, other data simulation settings, and various place holders. The true data generation model is expressed as:

$$Baby_{it} = \alpha_{b,i,t-1} Baby_{it} + \alpha_{bm} Mom_{i,t-1} + \zeta_{b,it}$$

$$Mom_{it} = \alpha_{m,i,t-1} Mom_{it} + \alpha_{mb} Baby_{i,t-1} + \zeta_{m,it}$$

$$\alpha_{m,it} = \alpha_{10} + \alpha_{11} x_{it}$$

$$\alpha_{b,it} = \alpha_{20} + \alpha_{21} x_{it}.$$

```
nt      <- 200 #Total number of time points
npad      <- 0 #Occasions to throw out to wash away the effects of initial condition
np        <- 50 #Total number of participants
ne        <- 4 #Number of latent variables
ny        <- 2 #Number of manifest variables
nx        <- 1 #Number of covariates
psi       <- matrix(c(1, .5, 0,0,  # Process noise variance-covariance matrix
                      .5, 1.2,0,0,
                      0,0,.0001,0,
                      0,0,0,.0001),
                    ncol = ne, byrow = T)
alpha10 = 0.5; alpha11 = .3
alpha20 = 0.4; alpha21 = .2

#Factor loading matrix
lambda    <- matrix(c(1,0,0,0,
                      0,1,0,0),
                    ncol = ne, byrow = TRUE)
theta     <- diag(.0001, ncol = ny, nrow = ny)  # Measurement error variances.
cr12 = 0
cr21 = -0.2
a0        <- c(0,0,alpha10,alpha20)
P0        <- matrix(c(1,0,0,0,
                      0,1,0,0,
                      0,0,.0001,0,
                      0,0,0,.0001),
                    ncol=ne,byrow=TRUE) #Initial covariance matrix for the trend elements
yall <- matrix(0,nrow = nt*np, ncol = ny+nx)
etaall <- matrix(0,nrow = nt*np, ncol = ne)
```

Data generation

Data generation starts here.

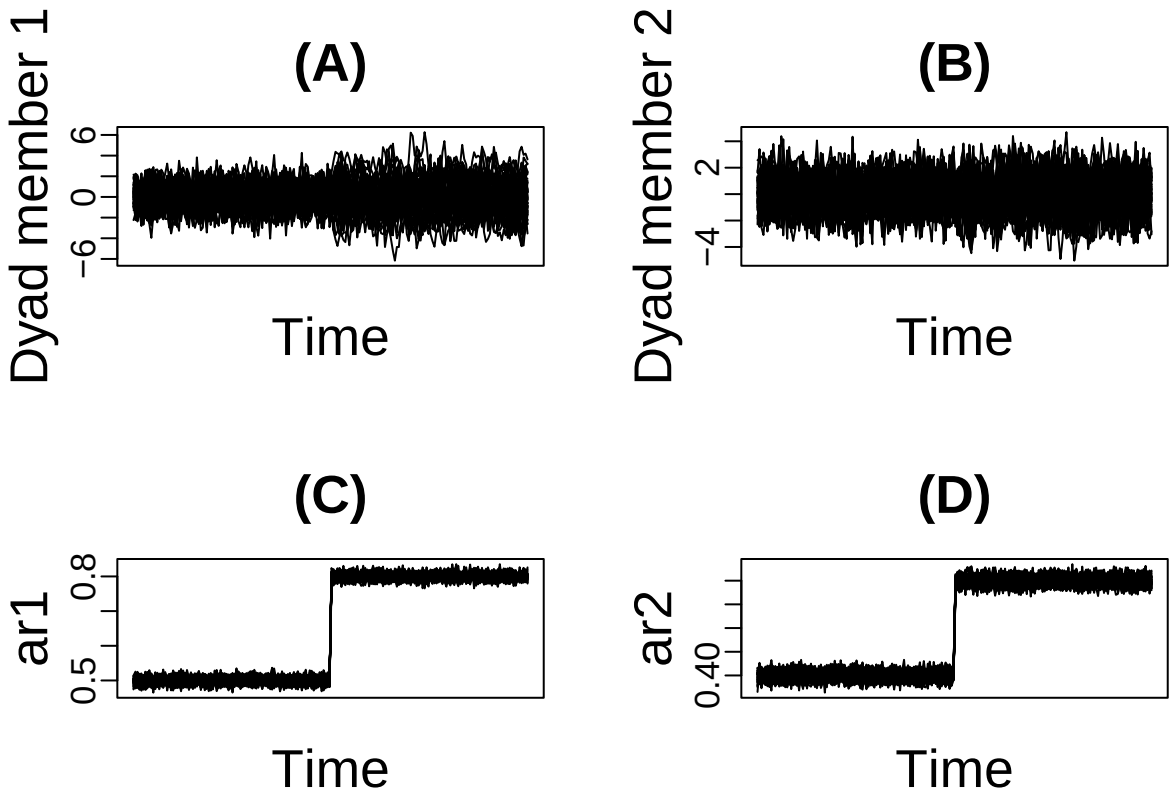
```
for (p in 1:np){
# Set up matrix for contemporaneous variables.
  etaC      <- matrix(0, nrow = ne, ncol = nt + npad)
  etaC[,1]  <- a0 + chol(P0)%*%rnorm(ne)

# Latent variable residuals.
  zeta      <- mvtnorm::rmvnorm(nt+npad, mean = rep(0,ne), sigma = psi)

# Measurement errors.
  epsilon   <- rmvnorm(nt+npad, mean = rep(0,ny), sigma = theta)

 TV_cov = c(rep(0,nt/2), rep(1,nt-nt/2))
  intercept = matrix(rep(0,ne),ncol=1)
# Generate latent factor scores
  for (i in 2:(nt+npad)){
    etaC[1,i] = etaC[3,i-1]*etaC[1,i-1] + cr12*etaC[2,i-1] + zeta[i,1]
    etaC[2,i] = etaC[4,i-1]*etaC[2,i-1] + cr21*etaC[1,i-1] + zeta[i,2]
    etaC[3,i] = alpha10 + alpha11*TV_cov[i] + zeta[i,3]
```

```
    etaC[4,i] = alpha20 + alpha21*TV_cov[i] + zeta[i,4]
   }#End of loop over nt

  eta <- t(etaC[,(npad+1):(npad+nt)])

  # generate observed series
  y   <- matrix(0, nrow = nt, ncol = ny)
  for (i in 1:nrow(y)){
    y[i, 1:ny] <- lambda %*% eta[i, ] + epsilon[i, ]
  }
  yall[(1+(p-1)*nt):(p*nt),1:ny] = y
  yall[(1+(p-1)*nt):(p*nt),(ny+1):(ny+nx)] = TV_cov
  etaall[(1+(p-1)*nt):(p*nt),] = eta
} #End of p loop

yall = cbind(rep(1:np,each=nt),rep(1:nt,np),yall)
etaall = cbind(rep(1:np,each=nt),rep(1:nt,np),etaall)
colnames(etaall) = c("ID","time",paste0("LV",1:ne))
etaall = data.frame(etaall)
colnames(yall) = c("ID","time",paste0("y",1:ny),paste0("cov",1:nx))
yall = data.frame(yall)
#Optionally: You could write out the data files and true latent variable scores
#file0 = paste0('TV_VAR.txt')
#file1 = paste0('TV_VARTrueLVs.txt')
#write.table(yall,file0,row.names=FALSE,col.names=FALSE,sep=",")
#write.table(etaall,file1,row.names=FALSE,col.names=FALSE,sep=",")
```

Plots of simulated data

Plots of the latent variables corresponding to baby, mom, and the TV AR coefficients.

```
par(mfrow=c(2,2))
par(cex.axis=1.3,cex.lab=2,mgp=c(2.0,.5,0),cex.main=2)
interaction.plot(etaall$time,etaall$ID,etaall$LV1,legend=F,lty=1,
                 ylab="Dyad member 1",xlab="Time",
                 main=("(A)"),xaxt = "n")

par(cex.axis=1.3,cex.lab=2,mgp=c(2.0,.5,0),cex.main=2)
interaction.plot(etaall$time,etaall$ID,etaall$LV2,legend=F,lty=1,
                 ylab="Dyad member 2",xlab="Time",
                 main=("(B)"),xaxt = "n")

par(cex.axis=1.3,cex.lab=2,mgp=c(2.0,.5,0),cex.main=2)
interaction.plot(etaall$time,etaall$ID,etaall$LV3,legend=F,lty=1,
                 ylab="ar1",
                 xlab="Time",main=("(C)"),xaxt = "n")

par(cex.axis=1.3,cex.lab=2,mgp=c(2.0,.5,0),cex.main=2)
interaction.plot(etaall$time,etaall$ID,etaall$LV4,legend=F,lty=1,
                 ylab="ar2",
                 xlab="Time",main=("(D)"),xaxt = "n")
```



Code for dynr model - Random walk model for the TV AR coefficients

Here we start preparing the "recipes'' for a dynr model. The TV AR coefficients are added to the latent variable vector. So now, there are four latent variables. In addition, we use a random walk model to approximate the over-time dynamics of the AR coefficients as:

$$\alpha_{m,it} = \alpha_{m,i,t-1} + \zeta_{\alpha m,it}$$

$$\alpha_{b,it} = \alpha_{b,i,t-1} + \zeta_{\alpha b,it}$$

$$\alpha_{b,it} = \alpha_{20} + \alpha_{21} x_{it}.$$

The random walk model is a convenient approximation model. It includes as a special case the time-invariant model, a model positing that a time-varying paramter (TVP) simply remains invariant over time. This is obtained when the process noise variances, $Var(\zeta_{\alpha b,it})$, and $Var(\zeta_{\alpha m,it})$, are equal to zero.

```
## Optimization function called.
## Starting Hessian calculation ...
## Finished Hessian calculation.
## Original exit flag:  3
## Modified exit flag:  3
## Optimization terminated successfully: ftol_rel or ftol_abs was reached.
## Original fitted parameters:  0.0015496 -0.20762 0.016238 0.50537 -0.044817
## -7.8231 -8.2473 0.53444 0.42636
##
## Transformed fitted parameters:  0.0015496 -0.20762 1.0164 0.51364 1.2158
## 0.00040039 0.00026196 0.53444 0.42636
##
## Doing end processing
```

```
## Successful trial
## Total Time: 1.9349
## Backend Time: 1.9232
```

```
## Coefficients:
##           Estimate Std. Error t value   ci.lower   ci.upper            Pr(>|t|)
## crmb     0.0015496  0.0081292    0.19 -0.0143834  0.0174826                0.42
## crbm    -0.2076235  0.0079620  -26.08 -0.2232286 -0.1920183 <0.0000000000000002
## zv_baby  1.0163706  0.0145808   69.71  0.9877929  1.0449484 <0.0000000000000002
## cov_bm   0.5136443  0.0124053   41.41  0.4893305  0.5379582 <0.0000000000000002
## zv_mom   1.2157533  0.0174096   69.83  1.1816312  1.2498755 <0.0000000000000002
## zv_arb   0.0004004  0.0000685    5.85  0.0002662  0.0005346 <0.0000000000000002
## zv_arm   0.0002620  0.0000641    4.09  0.0001364  0.0003876 <0.0000000000000002
## alpha10  0.5344414  0.0485133   11.02  0.4393572  0.6295257 <0.0000000000000002
## alpha20  0.4263635  0.0482835    8.83  0.3317296  0.5209973 <0.0000000000000002
##
## crmb
## crbm    ***
## zv_baby ***
## cov_bm  ***
## zv_mom  ***
## zv_arb  ***
## zv_arm  ***
## alpha10 ***
## alpha20 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## -2 log-likelihood value at convergence = 57049.63
## AIC = 57067.63
## BIC = 57132.52
```

Now fitting the true model for the TVPs

Now we fit the true model for the TVPs to the same data set. The time-varying covariate, x_{1it}, is included as a predictor in the dynamic functions for the TVPs.

```
##      [,1]                [,2]                [,3]
## [1,] "baby ~ baby ~ arb" "baby ~ mom ~ crmb" "baby ~ arb ~ baby"
## [2,] "mom ~ baby ~ crbm" "mom ~ mom ~ arm"   "mom ~ arb ~ 0"
## [3,] "arb ~ baby ~ 0"    "arb ~ mom ~ 0"     "arb ~ arb ~ 1"
## [4,] "arm ~ baby ~ 0"    "arm ~ mom ~ 0"     "arm ~ arb ~ 0"
##      [,4]
## [1,] "baby ~ arm ~ 0"
## [2,] "mom ~ arm ~ mom"
## [3,] "arb ~ arm ~ 0"
## [4,] "arm ~ arm ~ 1"
```

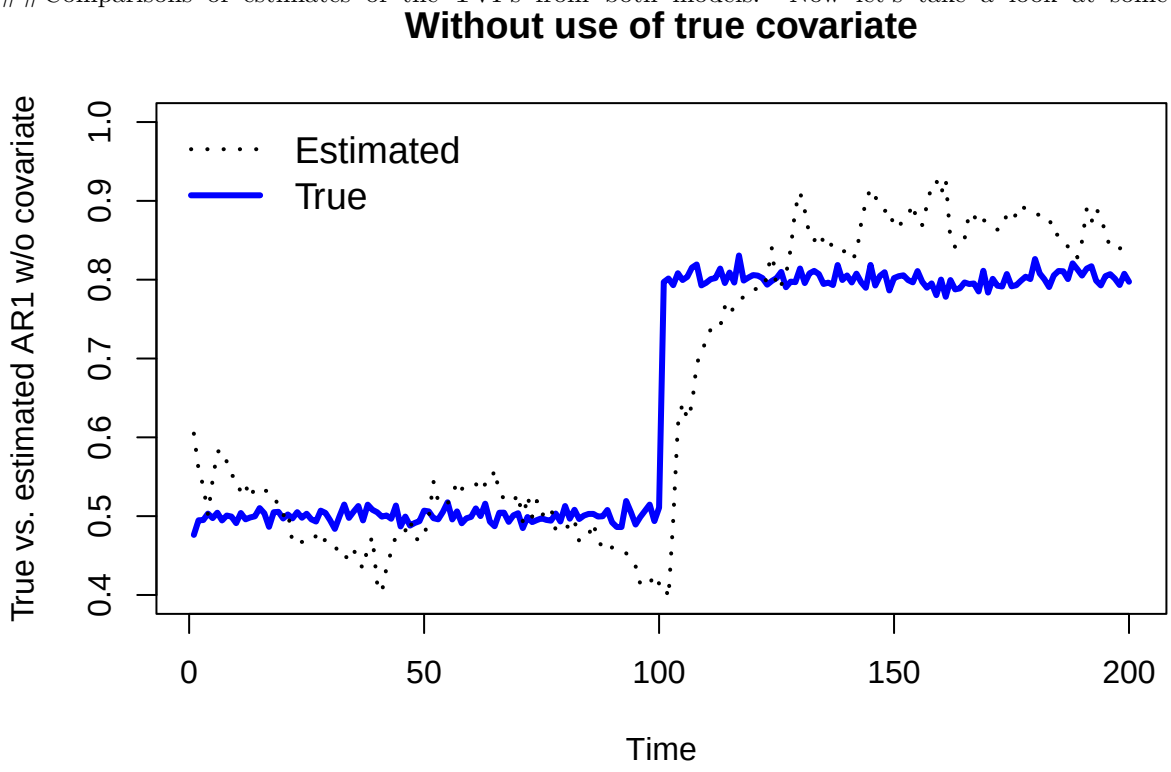
```
## arb
```

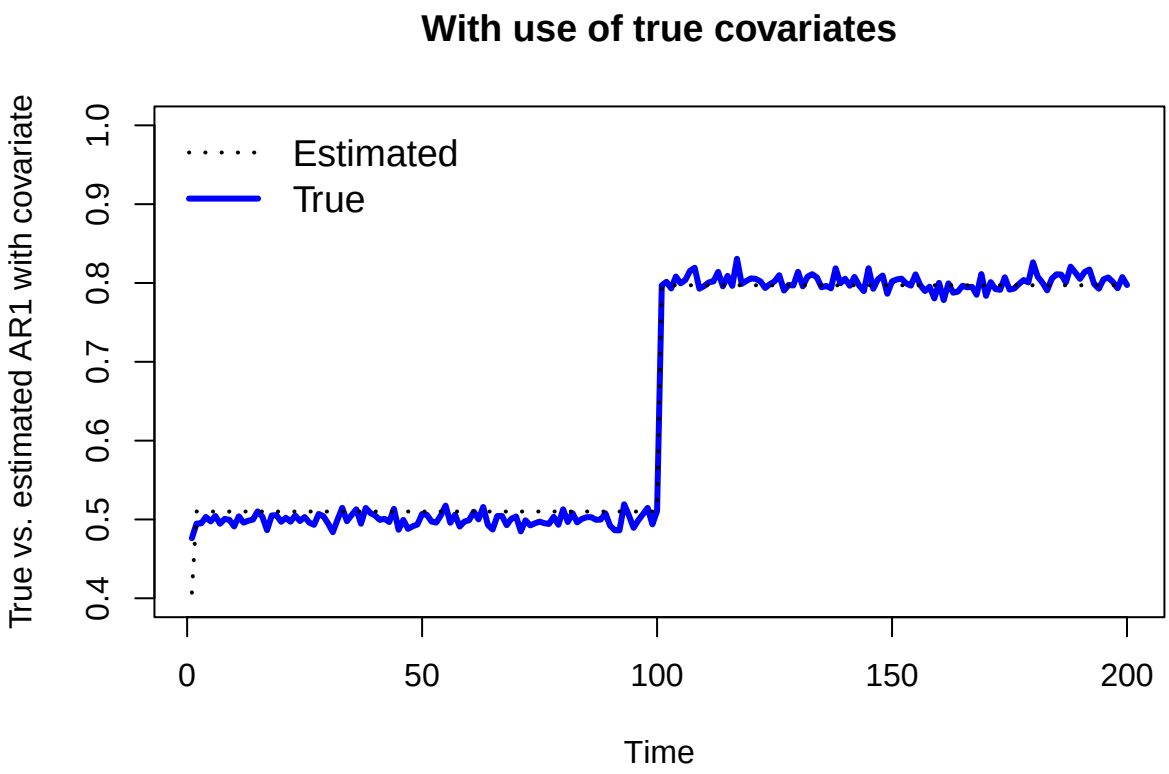
```
## Optimization function called.
## Starting Hessian calculation ...
```

```
## Finished Hessian calculation.
## Original exit flag:  3
## Modified exit flag:  3
## Optimization terminated successfully: ftol_rel or ftol_abs was reached.
## Original fitted parameters:  -0.00072322 -0.20105 0.51007 0.28703 0.39049
## 0.20803 0.011406 0.50401 -0.066673 -18.996 -4.3266 0.40709 0.35263 -22.337
## -1.6951
##
## Transformed fitted parameters:  -0.00072322 -0.20105 0.51007 0.28703 0.39049
## 0.20803 1.0115 0.50979 1.1924 0.0000000056266 0.013212 0.40709 0.35263
## 0.00000000019911 0.18359
##
## Doing end processing

## Warning in sqrt(diag(iHess)): NaNs produced

## Warning in sqrt(diag(x$inv.hessian)): NaNs produced

## Warning: These parameters may have untrustworthy standard errors: zv_arb,
## valpha1.

## Total Time: 5.1874
## Backend Time: 5.1745

## Coefficients:
##                 Estimate      Std. Error t value        ci.lower
## crmb     -0.000723215051  0.007906721040   -0.09 -0.016220103526
## crbm     -0.201045438007  0.007855917591  -25.59 -0.216442753551
## alpha10   0.510067136902  0.011737596550   43.46  0.487061870398
## alpha11   0.287030398833  0.013612485988   21.09  0.260350416556
## alpha20   0.390489310288  0.012676424439   30.80  0.365643974934
## alpha21   0.208030382912  0.015794678005   13.17  0.177073382874
## zv_baby   1.011471144929  0.014338111234   70.54  0.983368963304
## cov_bm    0.509787298632  0.012229631860   41.68  0.485817660641
## zv_mom    1.192437063313  0.019544393008   61.01  1.154130756918
## zv_arb    0.000000005627             NaN      NA             NaN
## zv_arm    0.013212073831  0.005887000816    2.24  0.001673764255
## alpha100  0.407086459508  0.143392908599    2.84  0.126041523016
## alpha200  0.352628598638  0.167969962065    2.10  0.023413522505
## valpha1   0.000000000199  0.000000060393    0.00 -0.000000118168
## valpha2   0.183585459952  0.183267318714    1.00 -0.175611884271
##                 ci.upper            Pr(>|t|)
## crmb      0.014773673424               0.464
## crbm     -0.185648122464 <0.0000000000000002 ***
## alpha10   0.533072403406 <0.0000000000000002 ***
## alpha11   0.313710381110 <0.0000000000000002 ***
## alpha20   0.415334645642 <0.0000000000000002 ***
## alpha21   0.238987382949 <0.0000000000000002 ***
## zv_baby   1.039573326555 <0.0000000000000002 ***
## cov_bm    0.533756936623 <0.0000000000000002 ***
## zv_mom    1.230743369708 <0.0000000000000002 ***
## zv_arb               NaN                   NA
## zv_arm    0.024750383407               0.012 *
```

```
## alpha100  0.688131396000              0.002 **
## alpha200  0.681843674771              0.018 *
## valpha1   0.000000118566              0.499
## valpha2   0.542782804174              0.158
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## -2 log-likelihood value at convergence = 56437.28
## AIC = 56467.28
## BIC = 56575.43
```

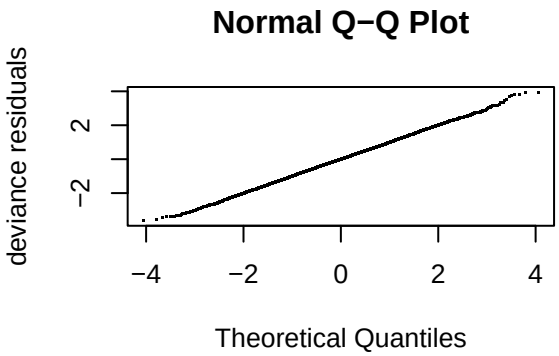
##Comparisons of estimates of the TVPs from both models. Now let's take a look at some plots!

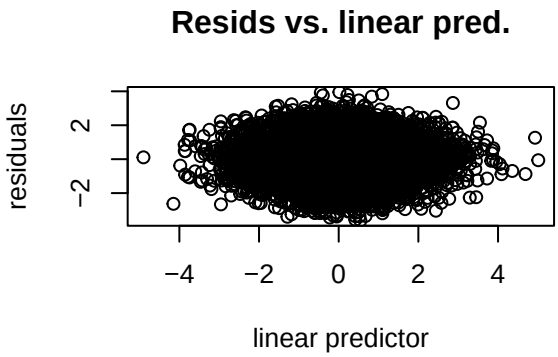


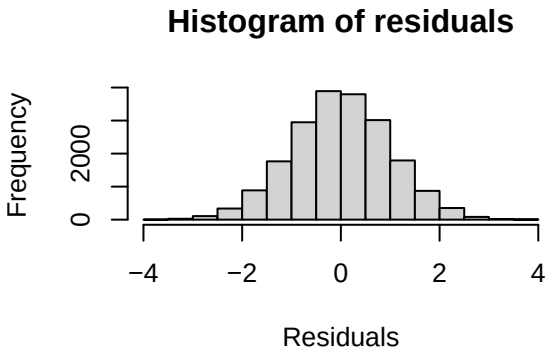


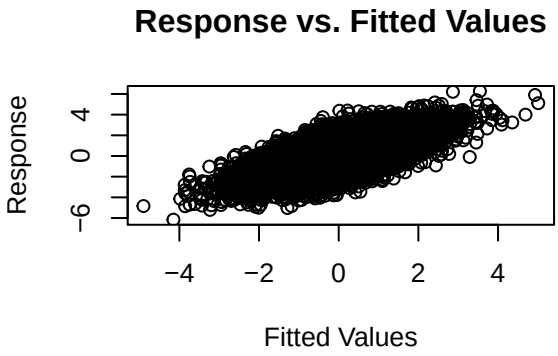
Generalized Additive Modeling (GAM) Approach

Here we consider an alternative approach, generalized additive modeling (GAM), which can be used to obtain smooth estimates of the trajectories of the TVPs via penalized splines. Here we use the gam function from the mgcv package.







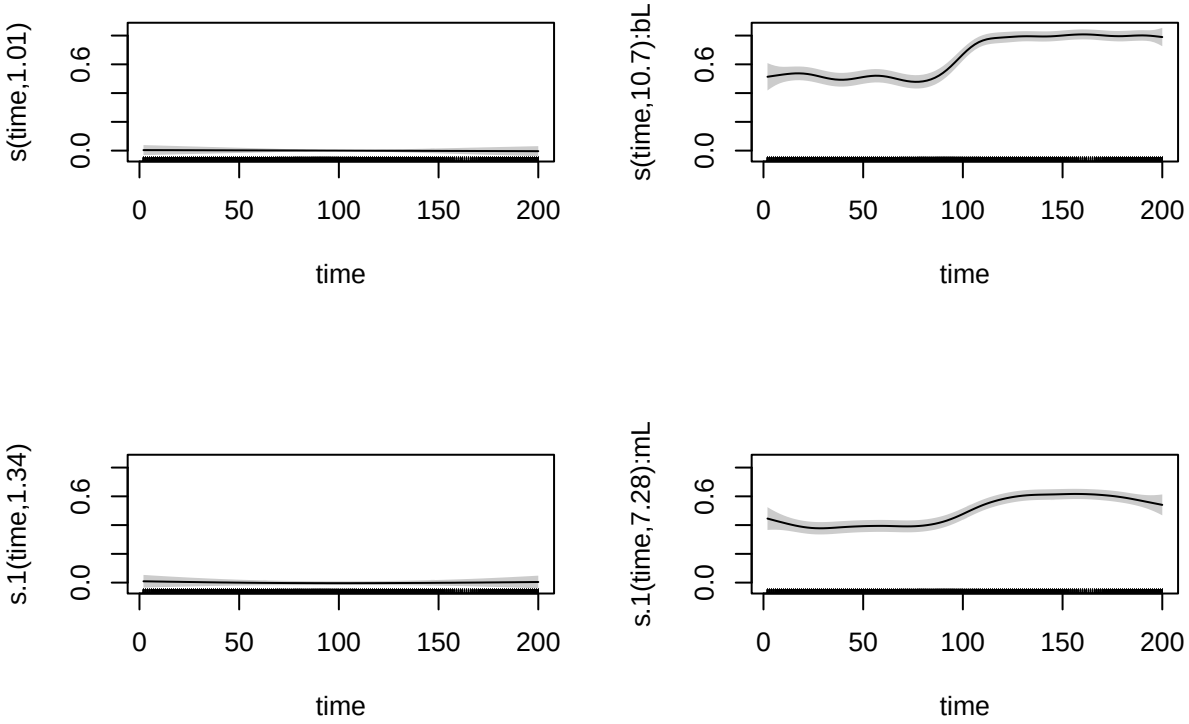


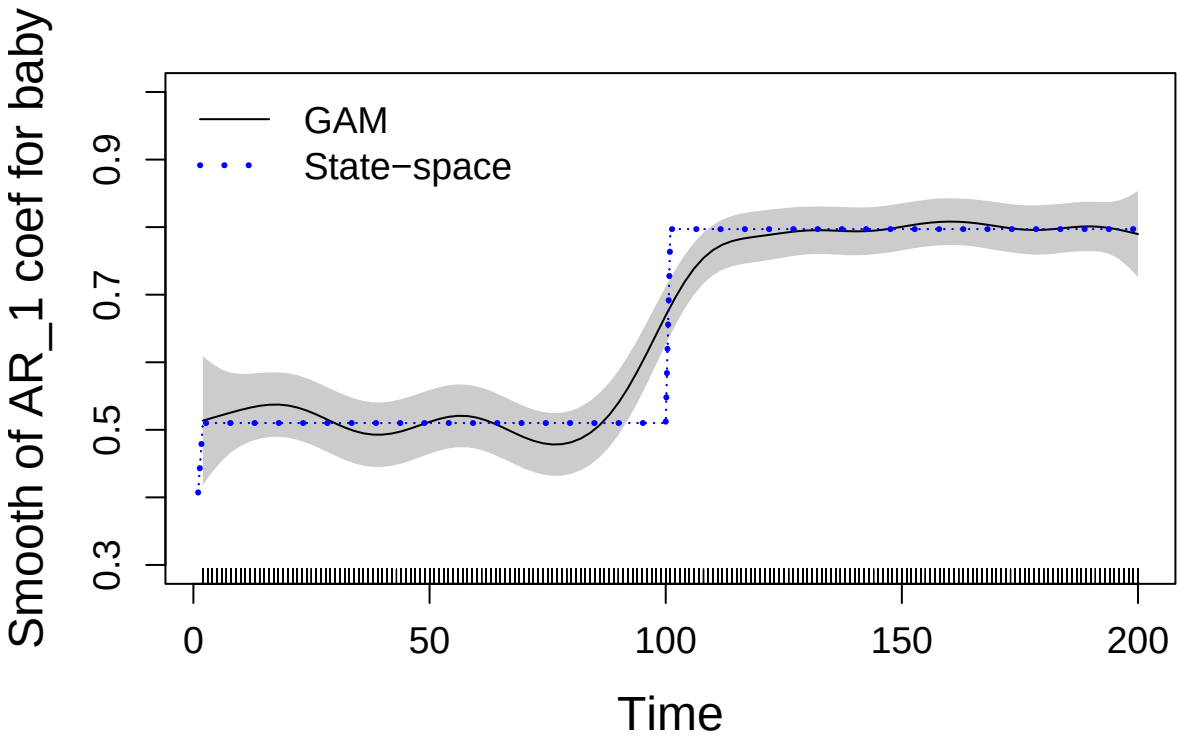

```
##
## Method: REML   Optimizer: outer newton
## full convergence after 14 iterations.
## Gradient range [-0.0005838,0.0052386]
## (score 9881.5 & scale 1).
## Hessian positive definite, eigenvalue range [0.0019863,1.5013].
## Model rank =  83 / 83
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                  k'   edf k-index p-value
## s(time)      19.00  1.01    1.02    0.93
## s(time):bL   20.00 10.70    1.02    0.92
## s.1(time)    19.00  1.34    1.02    0.95
## s.1(time):mL 20.00  7.28    1.02    0.96

##
## Family: Multivariate normal
## Link function:
##
## Formula:
## y1 ~ -1 + mL + s(time, k = 20) + s(time, by = bL, k = 20)
## y2 ~ -1 + s(time, k = 20) + s(time, by = mL, k = 20) + bL
##
## Parametric coefficients:
##      Estimate Std. Error z value            Pr(>|z|)
## mL   -0.00180    0.00791   -0.23                0.82
```

```
## bL.1 -0.20174    0.00787  -25.64 <0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                 edf Ref.df   Chi.sq             p-value
## s(time)       1.01   1.01     0.05                0.84
## s(time):bL   10.70  12.89 10262.84 <0.0000000000000002 ***
## s.1(time)     1.34   1.60     0.21                0.84
## s.1(time):mL  7.28   8.81  3738.59 <0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Deviance explained = 50.8%
## -REML = 9881.5  Scale est. = 1         n = 9950
```

```
##         [,1]    [,2]
## [1,] 1.01282 0.51099
## [2,] 0.51099 1.21729
```

Let's compare results from the state-space and GAM approach. The GAM approach provides a reasonable approximation of the TVP trajectories despite not using the time-varying covariate, x_{1it}. The TVP estimates from the two models are generally getting at slightly different things: the state-space approach estimates the trajectory of each individual's TVP over time; the GAM estimates provide an overall trajectory for each TVP characterizing *all individuals* over time. Their corresponding differences in standard error estimates reflect these differences. The confidence intervals for the TVP estimates from the state-space approach are very narrow in this case because the true mechanisms of change for the TVPs are deterministic, i.e., there are no process noises.




Smooth of AR_1 coef for baby
0.3
0.5
0.7
0.9
GAM
State−space
0
50
100
150
200
Time