RSNonlinearDiscreteExample

September 23, 2016

0.1 Introduction

This example illustrates how to fit a discrete-time regime-switching linear model in the `dynr` package.

0.2 Data

First, create a dynr data object using *dynr.data*. Here we have 6 observed variables.

```
require(dynr)


data(NonlinearDFAsim)
data <- dynr.data(NonlinearDFAsim, id="id", time="time",observed=colnames(NonlinearDFAsim)[c(3:8)])
```

0.3 Measurement Model

Next, we specify the measurement model using *prep.measurement*. The first three of the six observed variables load on the Positive Emotion (?) latent variable, and the last three load on the Negative Emotion variable. Parameters are indicated by parameter names (e.g. `lambda_*` in the following code), and fixed values are indicated by “fixed”. The `values.*` arguments specify the starting values and fixed values. The `params.*` arguments specify the free parameter names or the reserved word “fixed” for fixed parameters. Parameters with the same name are constrained to be equal. Since we do not have any covariate in this model. No `*.exo` arguments are supplied. Dispite this being a regime-switching model, we assume the same measurement model in the two regimes. Hence only one measurement model needs to be specified.

$$\begin{bmatrix} y1(t) \\ y2(t) \\ y3(t) \\ y4(t) \\ y5(t) \\ y6(t) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \lambda_{21} & 0 \\ \lambda_{31} & 0 \\ 0 & 1 \\ 0 & \lambda_{52} \\ 0 & \lambda_{62} \end{bmatrix} \begin{bmatrix} PE(t) \\ NE(t) \end{bmatrix} + \epsilon, \epsilon \sim N\Bigg(\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \epsilon_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \epsilon_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & \epsilon_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & \epsilon_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & \epsilon_5 & 0 \\ 0 & 0 & 0 & 0 & 0 & \epsilon_6 \end{bmatrix}$$

```
meas <- prep.measurement(
  values.load=matrix(c(1, .8, .8, rep(0, 3),
                       rep(0, 3), 1, .8, .8), ncol=2),
  params.load=matrix(c("fixed", "lambda_21", "lambda_31", rep("fixed",3),
                       rep("fixed",3), "fixed", "lambda_52","lambda_62"), ncol=2),
  state.names=c('PE', 'NE'))
```

0.4 Regime Probabilities

In the next step, we specify transition probability matrix between the regimes using *prep.regimes*. This transition probabilities from time *t* to time *t+1* are:

	Regime1(t+1)	Regime2(t+1)
Regime1(t)	$\frac{exp(p11)}{exp(p11) + exp(0)}$	$\frac{exp(0)}{exp(p11) + exp(0)}$
Regime2(t)	$\frac{exp(p21)}{exp(p21) + exp(0)}$	$\frac{exp(0)}{exp(p21) + exp(0)}$

(Here, $p11$ and $p21$ are model parameters. They can be perceived as odd ratios.)

```
regimes <- prep.regimes(
    values=matrix(c(.9, 0.1, 0.1, .9), 2, 2),
    params=matrix(c("p11", 0, 0, "p22"), 2, 2))
```

0.5 Dynamic Model

In the next chuck, we specify our dynamic models by first specifying the covariance matrices of measurement errors and dynamic noises using *prep.noise*, and then specifying the dynamic functions using *prep.formulaDynamics*. The dynamic models are:

We assume the same dynamic noise process applies to both regimes, hence we only have one matrix for dynamic noise specification (`*.latent`) in *prep.noise*. The `*.observed` arguments in *prep.noise* specify the measurement error (ϵ in the measurement model above).

```
mdcov <- prep.noise(
    values.latent=diag(0.3, 2),
    params.latent=diag(paste0("zeta_",1:2), 2),
    values.observed=diag(0.1, 6),
    params.observed=diag(paste0("epsilon_",1:6), 6))
```

We write dynamic functions in a list that contains two lists of functions, one for each regime.

```
formula=list(
  list(PE~a1*PE,
       NE~a2*NE),
  list(PE~a1*PE+c12*(exp(abs(NE)))/(1+exp(abs(NE)))*NE,
       NE~a2*NE+c21*(exp(abs(PE)))/(1+exp(abs(PE)))*PE)
)
```

Since this is a nonlinear model we need to specify the jacobian matrix containing the differentiation of each dynamic function above with respect to each latent variable (PE & NE). Although automatic differentiation is available for most other functions, the R package we use for automatic differentiation, Deriv, does not have automatic differentiation capability to handle the "abs" function used here.

```
jacob=list(
  list(PE~PE~a1,
       NE~NE~a2),
  list(PE~PE~a1,
       PE~NE~c12*(exp(abs(NE))/(exp(abs(NE))+1)+NE*sign(NE)*exp(abs(NE))/(1+exp(abs(NE))^2)),
       NE~NE~a2,
       NE~PE~c21*(exp(abs(PE))/(exp(abs(PE))+1)+PE*sign(PE)*exp(abs(PE))/(1+exp(abs(PE))^2))))
```

Then we combine dynamic functions and the jocobian matrix together with parameter specifications in *prep.formulaDynamics*.

```
dynm<-
prep.formulaDynamics(formula=formula,startval=c(a1=.3,a2=.4,c12=-.5,c21=-.5),isContinuousTime=FALSE,j
acobian=jacob)
```

The *prep.tfun* function is used here to transform regime switching probability parameters (odd ratios) to transition probabilities. This function can also be used to tranform parameters on a constrained scale to an unconstrained scale (e.g. exponential transformation to ensure parameters take positive values).

```
trans<-prep.tfun(formula.trans=list(p11~exp(p11)/(1+exp(p11)), p22~exp(p22)/(1+exp(p22))), formula.
inv=list(p11~log(p11/(1-p11)),p22~log(p22/(1-p22))), transCcode=FALSE)
```

0.6 Initial Values

After that, we specify values at time *t=0* using *prep.initial*. These values are used to initialize the recursive algorithm (extended Kalman filter) that dynr uses. The `*.inistate` arguments specify the initial (starting) states of the latent state variables. The `*.inicov` arguments specify the starting covariance matrix of the latent state variables. The `*.regimep` specifies the initial probabilities of the two regimes.

```
initial <- prep.initial(
    values.inistate=c(0, 0),
    params.inistate=c("fixed", "fixed"),
    values.inicov=diag(1, 2),
    params.inicov=diag("fixed", 2),
    values.regimep=c(.8, .2),
    params.regimep=c("fixed", "fixed")
)
```

0.7 Dynr Model

Now we put together everything we've previously specified in *dynr.model*. This code connects the recipes we've written up with our data and writes a c file in our working directory. We can inspect c functions that go with each recipe in the c file.

```
model <- dynr.model(dynamics=dynm, measurement=meas, noise=mdcov,
                    initial=initial, regimes=regimes, transform=trans,
                    data=data,
                    outfile="RSNonlinearDiscrete")
```

0.8 Tex Options

We can check our model specifications in a neatly printed pdf file using the following code. The printex() function produces a latex file with model specifications. The ParameterAs argument

```
printex(model,ParameterAs=model$param.names,printInit=TRUE, printRS=TRUE,
        outFile="RSNonlinearDiscrete.tex")
tools::texi2pdf("RSNonlinearDiscrete.tex")
system(paste(getOption("pdfviewer"), "RSNonlinearDiscrete.pdf"))
```

0.9 Optimization Step and Results

Finally, it is time to cook dynr (i.e. fit our model through parameter optimization)!

```
res <- dynr.cook(model)
```

And serve!

```
summary(res)
```

```
##                names parameters        s.e.   t-value    ci.lower
## a1                a1  0.2198461 0.019004976 11.567818  0.18259703
## a2                a2  0.2582208 0.019663032 13.132298  0.21968197
## c12              c12 -0.6848294 0.082305641 -8.320565 -0.84614553
## c21              c21 -0.6452563 0.087215512 -7.398412 -0.81619555
## lambda_21 lambda_21  1.1903216 0.023229644 51.241490  1.14479231
## lambda_31 lambda_31  1.1988357 0.023262089 51.536028  1.15324284
## lambda_52 lambda_52  1.0974432 0.019646210 55.860303  1.05893737
## lambda_62 lambda_62  0.9339182 0.017161902 54.418102  0.90028145
## zeta_1       zeta_1  0.3364646 0.014716377 22.863276  0.30762103
## zeta_2       zeta_2  0.2798348 0.010731100 26.076997  0.25880228
## epsilon_1 epsilon_1  0.2794314 0.008363511 33.410779  0.26303923
## epsilon_2 epsilon_2  0.1068529 0.006009409 17.780927  0.09507463
## epsilon_3 epsilon_3  0.0951393 0.005938563 16.020593  0.08349993
## epsilon_4 epsilon_4  0.1204069 0.004795519 25.108219  0.11100790
## epsilon_5 epsilon_5  0.1213492 0.005345796 22.699940  0.11087167
## epsilon_6 epsilon_6  0.1054444 0.004191905 25.154282  0.09722838
## p11              p11  0.9666511 0.011959637 80.826125  0.94321064
## p22              p22  0.8503182 0.043787588 19.419161  0.76449613
##             ci.upper
## a1         0.2570952
## a2         0.2967596
## c12       -0.5235133
## c21       -0.4743170
## lambda_21  1.2358508
## lambda_31  1.2444286
## lambda_52  1.1359491
## lambda_62  0.9675549
## zeta_1     0.3653082
## zeta_2     0.3008674
## epsilon_1  0.2958236
## epsilon_2  0.1186311
## epsilon_3  0.1067787
## epsilon_4  0.1298060
## epsilon_5  0.1318268
## epsilon_6  0.1136604
## p11        0.9900916
## p22        0.9361403
##
## -2 log-likelihood value at convergence = 28362.81
## AIC = 28398.81
## BIC = 28506.92
```