Web-based Supplementary Materials for "Assessing the impact of network data on the spatio-temporal spread of infectious diseases"

Birgit Schrödle, Leonhard Held and Håvard Rue

1 Web Appendix A

Univariate time series

First it is shown, how model PM_3 from Section 3.2 is coded in `INLA`. For a description of the seasonal term see equation (2). The Salmonella agona data are available in the `R` package `surveillance`.

```
> ### see Section 3.2
> ### PM_3: autoregression - yes, seasonality - yes
> # load salmonella agona data
> library(INLA)
> library(surveillance)
> data(salmonella.agona)
> Y <- salmonella.agona$observed
> T <- length(Y)
> t <- 1:T
> p <- 52
> sinterm <- sin(2*pi*t/p)
> costerm <- cos(2*pi*t/p)
> zeta <- t
> #
> # define model formula
> f_pm3 <- Y~1+f(zeta,model="ar1",param=c(0.1,0.001,0,0.2))+t+sinterm+costerm
> #
> # call model
> m_pm3 <- inla(f_pm3,family="Poisson",data=data.frame(Y,zeta,t,sinterm,costerm))
```

Multivariate time series

The implementation of the multivariate time series model (4) in `R` is similar to the univariate case. See as an application the code for model PM_1 for the Coxiellosis data (see Section 4.2). Here, the important feature is the `replicate`-option, which fits identical copies of a temporal AR(1) process for all regions.

```
> # Coxiellosis data are stored in a data.frame data_cox<-cbind(Y,E)
> # the ordering of the data is (i,t)
> Y <- data_cox[,1]
> E <- data_cox[,2]  # offset m_it
> #
> I <- 185
> T <- 6
> #
> zeta <- rep(1:T,I)
> id <- rep(1:I,each=T)
> #
> # define model formula for model: PM_3
> f_pm3 <- Y~f(zeta,model="ar1",replicate=id,param=c(0.1,0.001,0,0.2))
> #
> # call model
> m_pm3 <- inla(f_pm3,family="Poisson",E=E,data=data.frame(Y,E,zeta,id))
```

Multivariate time series models including a ρ component

If a ρ component as in equation (6) should be included, an augmented model containing pseudo-observations must be implemented. A detailed description of the available toolbox is given in Ruiz-Cardenas et al. (2010). To code the variable and weight vectors for the first and second stage of model (6) an automatic `coding` function can be used.

```
> # function to code the variable and weight vectors/matrices
> # of the components of the first and second stage of model (6)
> # the matrix wm contains the weights (w_ji)^T
> #
> coding<-function(I,T,wm){
+ # function to transform the (i,t) coordinates of an observation in a vector coordinate j
+ idx <- function(t,i,T){
+ j <- (i-1)*T+t
+ if (t<1) return(NA)
+ else return(j)
+ }
+ #
+ # initialize the second part of the variable vector for zeta and zetatm1 (lambda)
+ zeta.p2 <- rep(NA,I*T)
+ zetatm1.p2 <- rep(NA,I*T)
+ #
+ # initialize the second part of the rho component matrix and the respective weights
+ # the rho matrix contains one column for each region
+ rho.p2 <- matrix(NA,nrow=I*T,ncol=I)
+ w.rho.p2 <- matrix(NA,nrow=I*T,ncol=I)
+ #
+ # go through the vector with dimension I*T=J
+ j <- 1
+ for (i in 1:I){
+ for (t in 1:T){
+ # define the coordinates of zeta and zetatm1 for each vector node j
+ zeta.p2[j] <- idx(t,i,T)
+ zetatm1.p2[j] <- idx(t-1,i,T)
+ #
+ # define the rho coordinate and weight of each region k for vector node j
+ for (k in 1:I){
+ rho.p2[j,k] <- ifelse(wm[i,k]==0,NA,idx(t-1,k,T))
+ w.rho.p2[j,k] <- ifelse(wm[i,k]==0,NA,-wm[i,k])
+ }
+ j <- j+1}}
+ #
+ # put together the first and second part of the variable/weight vectors and matrices
+ zeta <- c(zeta.p2,zeta.p2)
+ zetatm1 <- c(rep(NA,I*T),zetatm1.p2)
+ rho <- rbind(matrix(NA,nrow=(I*T),ncol=I),rho.p2)
+ w.rho <- rbind(matrix(NA,nrow=(I*T),ncol=I),w.rho.p2)
+ #
+ return(list(zeta,zetatm1,rho,w.rho))}
```

The augmented model can be fitted using two different likelihoods for the real (Poisson) and pseudo-observations (Gaussian). The important feature here is the `same.as`-option, which allows to have the same ρ for each region.

```
> # set up the Y matrix (2*I*Tx2) with pseudo-observations for the evolution equation
> Y <- matrix(NA,nrow=(T+T)*I,2)
> Y[1:(I*T),1] <- data_cox[,1]
> Y[((I*T)+1):(2*T*I),2] <- 0
> #
> # code the variable vector for components on the first stage - here: alpha
```

```
> alpha <- c(rep(1,T*I),rep(NA,T*I))
> #
> # code the variable vectors for components on the first/second stage - here: zeta, zetatm1, rho
> # run function coding - wm can be a neighbourhood or other weight matrix
> code2ndstage <- coding(I,T,wm)
> # variable vector for zeta
> zeta <- code2ndstage[[1]]
> # variable vector for zetatm1
> zetatm1 <- code2ndstage[[2]]
> # weights for zetatm1
> w.zetatm1 <- ifelse(is.na(zetatm1)==TRUE,NA,-1)
> # variable matrix for rho
> rho <- code2ndstage[[3]]
> # weight matrix for rho
> w.rho <- code2ndstage[[4]]
> #
> # initialize the model formula
> # use the "copy" feature to obtain an identical copy of the zeta process
> # to estimate scaling parameters lambda (for zetatm1) and rho (for rho.1, rho.2,...)
> # set the option fixed=FALSE
> f <- "Y~f(zeta,model=\"iid\",fixed=TRUE,initial=-10)+
+                       +f(zetatm1,w.zetatm1,copy=\"zeta\",fixed=FALSE,param=c(0,1))+
+                       +f(rho.1,w.rho.1,copy=\"zeta\",fixed=FALSE,param=c(0,1))+alpha-1"
> #
> # or for a Fisher z-transformed lambda and rho (range-option)
> # f <- "Y~f(zeta,model=\"iid\",fixed=TRUE,initial=-10)+
> #         +f(zetatm1,w.zetatm1,copy=\"zeta\",fixed=FALSE,param=c(0,0.2),range=c(-1,1))+
> #         +f(rho.1,w.rho.1,copy=\"zeta\",fixed=FALSE,param=c(0,0.2),range=c(-1,1))+alpha-1"
> #
> # split the weight matrix for rho in separate vectors for each region - here: region 1
> vn <- paste("rho",1,sep=".")
> assign(vn,rho[,1])
> vn.w<-paste("w.rho",1,sep=".")
> assign(vn.w,w.rho[,1])
> #
> # run a split loop over all columns (regions) of the variable and weight matrix of rho
> for (i in 2:I){
+ vn <- paste("rho",i,sep=".")
+ assign(vn,rho[,i])
+ vn.w <- paste("w.rho",i,sep=".")
+ assign(vn.w,w.rho[,i])
+ #
+ # add a rho term for each region to the formula object
+ # to make sure that it's the same rho for each region, use option "same.as"
+ f <- paste(f,"+f(rho.",i,",w.rho.",i,",copy=\"zeta\",same.as=\"rho.1\")",sep="")}
> #
> # convert the formula in a formula object
> f_pm4 <- as.formula(f)
> #
> # define the offset E - on the first stage
> E <- c(data_cox[,2],rep(NA,I*T))
> #
> # call model - with two different likelihoods for the data and the pseudo-observations
> m_pm4 <- inla(f_pm4,family=c("Poisson","Gaussian"),data=data.frame(Y),
+               E=E,control.data=list(list(),list(initial=0)))
```

References

Ruiz-Cardenas, R., Krainski, E. T., and Rue, H. (2010). Fitting dynamic models using integrated nested Laplace approximations - INLA. Technical report, NTNU Trond-

heim, Norway.