2021 금융위험관리 중간과제

202STG01 고유정

2021 5 22

첨부파일의 repeated_data_management를 사용하여 mydata를 추출하세요. 이 데이터를 SSM모델을 통해 설명하려고 합니다.

n ~TypeCity+TypeCounty+TypeMisc+TypeSchool+TypeTown+factor(col.Cov.Idx)

예를 들어 다음과 같은 모델 Estimation결과를 얻으려고 합니다.

Table 4: Estimation result

Parameter	Est	Std.error	95% CI		
Fixed effect					
(Intercept)	-3.844	0.464	-4.783	-2.988	*
TypeCity	0.955	0.485	-0.042	1.934	
TypeCounty	2.280	0.476	1.322	3.197	*
TypeSchool	0.786	0.451	-0.036	1.709	
TypeTown	-1.097	0.546	-2.171	-0.027	*
TypeVillage	0.326	0.458	-0.568	1.271	
Coverage2	1.474	0.235	1.022	1.934	*
Coverage3	2.339	0.261	1.828	2.887	*
Random effect					
ϕ	0.952	0.019	0.909	0.982	*
σ	0.307	0.058	0.193	0.421	*

또한 이 결과를 사용하여 data.valid 의 n에 대해 예측해 보려고 합니다. 다음 질문에 답하세요.

1> data.train 과 data.valid 를 다음과 같은 형태를 가진 data.frame 으로 변환하세요.

ID	N1	···	N5	N 6 (test)	X0	X1	···	Xp	
xxxx	0		NA	1	1	0	1	1	
yyyy	NA	1	1	0	1	0	1	0	
					1	1	0	2	
					1	1	0	2	

*단 여기서 설명변수 X는 시간에 따라 변하지 않는 것으로 가정하고 t=1시점의 X정보를 가져 옵니다.

```
# ========= Problem1 =========
source("repeated_data_management.R")
```

```r
## dataframe ##
res1 = mydata[which(mydata$PolicyNum == unique(mydata$PolicyNum)[1]),][1,]
for (i in 2:length(unique(mydata$PolicyNum))){
  sam = mydata[which(mydata$PolicyNum == unique(mydata$PolicyNum)[i]),]
  row = sam[which(sam$Year==min(sam$Year)),]
  res1 = rbind(res1, row)
}
res1 = res1[,-2] # remove Year
head(res1)
```

```
##    PolicyNum  ClaimBC  ClaimIM  ClaimPN  ClaimPO ClaimCN  ClaimCO TypeCity
## 1     120002     0.00     0.00     0.00     0.00    0.00  6275.06        0
## 6     120003     0.00 21589.93     0.00     0.00    0.00  4206.64        0
## 11    120004 13003.08     0.00 22405.56  2712.53 1032.26     0.00        0
## 16    120005     0.00  2775.65  3834.33     0.00 1722.52 38167.20        0
## 21    120008 48060.82     0.00  4235.19     0.00    0.00 41587.00        0
## 31    120010 20110.06   796.90  7369.74 32776.10  780.82  1589.80        0
##    TypeCounty TypeMisc TypeSchool TypeTown TypeVillage IsRC CoverageBC
## 1           1        0          0        0           0    1   22.71446
## 6           1        0          0        0           0    1   99.35338
## 11          1        0          0        0           0    1   25.87963
## 16          1        0          0        0           0    1   31.46258
## 21          1        0          0        0           0    1   46.05153
## 31          1        0          0        0           0    1   71.01750
##    lnDeductBC NoClaimCreditBC    yAvgBC FreqBC CoverageIM lnDeductIM
## 1    6.907755               0     0.000      0   3.430260   6.907755
## 6    8.517193               0     0.000      0   9.198051   8.517193
## 11   6.214608               0  4334.360      3   4.055599   6.214608
## 16   6.907755               0     0.000      0   4.633063   6.214608
## 21   6.214608               0 16020.273      3   2.754886   6.214608
## 31  10.126631               0  5027.515      4   4.089611   8.517193
##    NoClaimCreditIM   yAvgIM FreqIM CoveragePN NoClaimCreditPN    yAvgPN Fr
eqPN
## 1                0     0.00      0   0.724329               0    0.0000
   0
## 6                0 10794.97      2   1.192516               0    0.0000
   0
## 11               0     0.00      0   0.591822               0 5601.3900
   4
## 16               0  2775.65      1   1.099368               0  958.5825
   4
## 21               0     0.00      0   0.943535               0 1411.7300
   3
## 31               0   398.45      2   0.444427               0  921.2175
   8
##    CoveragePO NoClaimCreditPO    yAvgPO FreqPO CoverageCN NoClaimCreditCN
## 1    1.692349               0    0.0000      0   0.724329               0
## 6    4.087988               0    0.0000      0   1.192516               0
## 11   0.518618               0  904.1767      3   0.591822               0
```

```
## 16   2.900052               0    0.0000      0   1.099368               0
## 21   1.091113               0    0.0000      0   0.943535               0
## 31   4.262824               0 8194.0250      4   0.444427               0
##     yAvgCN FreqCN CoverageCO NoClaimCreditCO    yAvgCO FreqCO col.freq  co
l.Cov
## 1     0.00      0   1.831077               0  3137.530      2        2 2.5
55406
## 6     0.00      0   4.034988               0  1402.213      3        3 5.2
27504
## 11 1032.26      1   0.518618               0     0.000      0        1 1.1
10440
## 16 1722.52      1   2.900052               0 38167.200      1        2 3.9
99420
## 21    0.00      0   1.091113               0 41587.000      1        1 2.0
34648
## 31  390.41      2   4.262824               0   794.900      2        4 4.7
07251
##    col.Cov.Idx n        s
## 1            2 2  6275.06
## 6            2 3  4206.64
## 11           1 1  1032.26
## 16           2 2 39889.72
## 21           2 1 41587.00
## 31           2 4  2370.62
```

```
res2 = as.data.frame(acast(data=mydata, PolicyNum ~ Year, value.var='n', fill
=NA))
res2$PolicyNum = unique(mydata$PolicyNum)
res2 = merge(res2, data.valid[,c('PolicyNum', 'n')], by='PolicyNum', all=TRU
E)
colnames(res2) = c('PolicyNum', paste0('n_', seq(2006,2011)))
head(res2)
```

```
##   PolicyNum n_2006 n_2007 n_2008 n_2009 n_2010 n_2011
## 1    120002      2      0      4      2      0      1
## 2    120003      3      4      3      3      4      3
## 3    120004      1      2      0      4      4      1
## 4    120005      2      4      6      1      3      1
## 5    120008      1      4      3      1      1      1
## 6    120010     NA     NA     NA      4      3      3
```

```
data = merge(res2, res1, by='PolicyNum')
final_data = subset(data, select = c('PolicyNum', 'n_2006', 'n_2007', 'n_2008
', 'n_2009', 'n_2010', 'n_2011',
                                     'TypeCity', 'TypeCounty', 'TypeSchool',
'TypeTown', 'TypeVillage',
                                     'col.Cov.Idx'
))
```

```
## Result
head(final_data)
```

```
##   PolicyNum n_2006 n_2007 n_2008 n_2009 n_2010 n_2011 TypeCity TypeCounty
## 1    120002      2      0      4      2      0      1        0          1
## 2    120003      3      4      3      3      4      3        0          1
## 3    120004      1      2      0      4      4      1        0          1
## 4    120005      2      4      6      1      3      1        0          1
## 5    120008      1      4      3      1      1      1        0          1
## 6    120010     NA     NA     NA      4      3      3        0          1
##   TypeSchool TypeTown TypeVillage col.Cov.Idx
## 1          0        0           0           2
## 2          0        0           0           2
## 3          0        0           0           1
## 4          0        0           0           2
## 5          0        0           0           2
## 6          0        0           0           2
```

2> JAGS Program 을 사용하여 다음 Poisson-AR(1) SSM 모델의 parameter 들 $(\phi, \sigma^2, \beta_0, \cdots, \beta_p)$의 사후 분포를 구하고 각 parameter들의 95% confidence를 구하세요.

$$N_t | R_t \sim Pois(\lambda_t \exp(R_t))$$

$$R_t = \phi * R_{t-1} + \epsilon_t$$

$$\text{where } \epsilon_t \sim N(0, \sigma^2) \text{ and } R_0 \sim N\left(0, \frac{\sigma^2}{1-\phi}\right).$$

이때 N의 경우 t=1, …, 5까지만 사용합니다. t=6일 경우 Test set 으로 남겨 놓습니다.
Hint: 주어진 JAGS프로그램을 참조할수 있음.

```
# ========= Problem2 =========
## SSM-Pois-AR(1)
X <- unname(as.matrix(final_data[8:12]))
X <- cbind(matrix(rep(1, dim(X)[1])), X)
N <- unname(as.matrix(final_data[,2:6]))
TT_N <- apply(!is.na(N), 1, sum)

# T1: indicator matrix that indicates the year of non NA N
T1 <- matrix(nrow=nrow(N), ncol=5)
for(i in 1:nrow(N)){
  ind <- which(!is.na(N[i,]))
  ind <- append(ind, rep(NA, 5-length(ind)))
  T1[i,] <- ind
}

head(T1)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    1    2    3    4    5
## [3,]    1    2    3    4    5
## [4,]    1    2    3    4    5
## [5,]    1    2    3    4    5
## [6,]    4    5   NA   NA   NA
```

```
TTT = rep(5, dim(X)[1])

# datalist
dataList=list(N=N, X=X, TTT=TTT, TT_N=TT_N, T1=T1,
              m.beta=c(-3, rep(0,dim(X)[2]-1)), # initial value
              inv_Sigma=1*diag(dim(X)[2]),
              c0=.001, d0=0.001, K = dim(X)[1] # hyperparameter
)

#initial value for the parameters
init.beta= c(-3, rep(0,dim(X)[2]-1))
init.phi = 0.8; init.sig = 0.5

# model
modelString="model {

####### Prior ########
for(i in 1:K){

    R[i,1] ~ dnorm(0, (1-phi^2)/sigsq)

    for(t in 2:(TTT[i]+2)){ #TTT = 5 #number of years regardless of observed
or not.
        R[i,t] ~ dnorm(phi*R[i,t-1], 1/sigsq)
    }
    return_hidden_1[i] = R[i,TTT[i]+2]
}

####### Likelihood Part #######
for(i in 1:K){ #K: number of people
    for(t in 1:TT_N[i]){ #TT_N[i]=3: number of total observed years = 3 / 5
                         #T1[i,]= [1,3,5,NA, NA] # indication of observed years
        mu_N[i,T1[i,t]] = exp( inprod(X[i,],beta_hat[])) * exp(R[i,T1[i,t]+
1])
        N[i,T1[i,t]] ~ dpois( mu_N[i,T1[i,t]] )
    }
}

####### prior #######
beta_hat[1:length(m.beta)] ~ dmnorm(m.beta[], inv_Sigma)
```

```
invsigsq ~ dgamma(c0,d0)
sigsq = 1/invsigsq
phi ~ dnorm(0, 1e-4)T(-1,1)

}
"
# run jags
inits1=list(beta=init.beta, sig=init.sig,
            phi=init.phi, #sig0=init.sig,
            .RNG.name="base::Super-Duper")
inits2=list( beta=init.beta, sig=init.sig,
             phi=init.phi, #sig0=init.sig,
             .RNG.name="base::Wichmann-Hill")
inits3=list(beta=init.beta, sig=init.sig,
            phi=init.phi, #sig0=init.sig,
            .RNG.name="base::Mesenne-Twister")

nChains=3; nAdapt=5000; nUpdate=30000; nSamples=30000; nthin=5
ptm.init <- proc.time()
runJagsOut = run.jags(method="parallel", model=modelString,
                      monitor=c("beta_hat", "sigsq", "phi"),
                      data=dataList, inits=list(inits1, inits2, inits3),
                      n.chains=nChains, adapt=nAdapt, burnin=nUpdate,
                      sample=ceiling(nSamples/nChains), thin=nthin,
                      summarise=TRUE, plots=TRUE)

## Calling 3 simulations using the parallel method...
## Following the progress of chain 1 (the program will wait for all chains
## to finish before continuing):
## Welcome to JAGS 4.3.0 on Sat May 22 15:34:09 2021
## JAGS is free software and comes with ABSOLUTELY NO WARRANTY
## Loading module: basemod: ok
## Loading module: bugs: ok
## . . Reading data file data.txt
## . Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 1467
##    Unobserved stochastic nodes: 2866
##    Total graph size: 14938
## . Reading parameter file inits1.txt
##
## WARNING: Unused variable(s) in initial value table:
## beta
## sig
##
## . Initializing model
## . Adapting 5000
```

```
## -------------------------------------------------| 5000
## ++++++++++++++++++++++++++++++++++++++++++++++++++ 100%
## Adaptation successful
## . Updating 30000
## -------------------------------------------------| 30000
## ************************************************** 100%
## . . . . Updating 50000
## -------------------------------------------------| 50000
## ************************************************** 100%
## . . . . Updating 0
## . Deleting model
## .
## All chains have finished
## Simulation complete.  Reading coda files...
## Coda files loaded successfully
## Calculating summary statistics...
## Calculating the Gelman-Rubin statistic for 8 variables....
## Finished running the simulation
```

```
summary(runJagsOut)[,c(1,3)]  # 신뢰구간
```

```
##                Lower95   Upper95
## beta_hat[1] -2.2775800 -1.090660
## beta_hat[2]  0.2985640  1.642070
## beta_hat[3]  1.8338800  3.127470
## beta_hat[4] -0.5821710  0.690418
## beta_hat[5] -1.9115100 -0.329270
## beta_hat[6] -0.6973360  0.689728
## sigsq        0.0231434  0.161005
## phi          0.9222320  0.989543
```

3> 우선 2>의 결과를 이용하여 SSM 모델의 parameter 들을 $\hat{\phi}, \hat{\sigma^2}, \hat{\beta_0}, \cdots, \hat{\beta_p}$을 사후분포의 median 값으로 고정시킵니다. 다음으로 2>에서 JAGS Program을 수정하여 $R[i,t+1]$의 사후 분포를 구하고 이를 이용하여 다음 형태의 예측을 하세요.

$$\hat{E_1}[N_{i,t+1}|n_{i,1},\cdots,n_{i,t}] = \frac{\sum_{j=1}^{J} N_{i,t+1}^{(j)}}{J}$$

그리고, 다음과 같이 정의된 MSE와 MAE를 정의합니다.

$$MSE = \frac{\sum_{i=1}^{K}(N_{i,t+1} - \hat{E_1}[N_{i,t+1}|n_{i,1},\cdots,n_{i,t}])^2}{K}$$

and

$$MAE = \frac{\sum_{i=1}^{K}|N_{i,t+1} - \hat{E_1}[N_{i,t+1}|n_{i,1},\cdots,n_{i,t}]|}{K}$$

이를 사용하여 실제 data.train과 data.valid를 사용한 Test MSE와 Test MAE 를 계산하세요.

```
# ========= Problem3 =========
post_beta = summary(runJagsOut)[1:6,2]
post_sigsq = summary(runJagsOut)[7,2]
post_phi = summary(runJagsOut)[8,2]

# datalist
dataList=list(N=N, X=X, TTT=TTT, TT_N=TT_N, T1=T1,
              beta=post_beta, sigsq=post_sigsq,
              phi=post_phi, K = dim(X)[1] # hyperparameter
)

# model
modelString="model {

######## Prior ########
for(i in 1:K){
    R[i,1] ~ dnorm(0, (1-phi^2)/sigsq)
    for(t in 2:(TTT[i]+2)){ #TTT = 5 #number of years regardless of observed
or not.
        R[i,t] ~ dnorm(phi*R[i,t-1], 1/sigsq)
    }
    return_hidden_1[i] = R[i,TTT[i]+2] # R[i,t+1]
}

######## Likelihood Part ########
for(i in 1:K){ #K: number of people
    for(t in 1:TT_N[i]){ #TT_N[i]=3: number of total observed years = 3 / 5
                         #T1[i,]= [1,3,5,NA, NA] # indication of observed years
        mu_N[i,T1[i,t]] = exp( X[i,] %*% beta[]) * exp(R[i,T1[i,t]+1])
```

```
        N[i,T1[i,t]] ~ dpois( mu_N[i,T1[i,t]] )
    }

}
}
"

# run jags
inits1=list(.RNG.name="base::Super-Duper")
inits2=list(.RNG.name="base::Wichmann-Hill")
inits3=list(.RNG.name="base::Mesenne-Twister")
nChains=3; nAdapt=5000; nUpdate=30000; nSamples=30000; nthin=5
ptm.init <- proc.time()
runJagsOut1 = run.jags(method="parallel", model=modelString,
                       monitor=c("return_hidden_1", "return_mu_N"),
                       data=dataList,
                       n.chains=nChains, adapt=nAdapt, burnin=nUpdate,
                       sample=ceiling(nSamples/nChains), thin=nthin,
                       summarise=TRUE, plots=TRUE)

## Warning: No initial values were provided - JAGS will use the same initial
values
## for all chains

## Warning: You attempted to start parallel chains without setting different
PRNG
## for each chain, which is not recommended. Different .RNG.name values have
been
## added to each set of initial values.

## Calling 3 simulations using the parallel method...
## Following the progress of chain 1 (the program will wait for all chains
## to finish before continuing):
## Welcome to JAGS 4.3.0 on Sat May 22 15:45:52 2021
## JAGS is free software and comes with ABSOLUTELY NO WARRANTY
## Loading module: basemod: ok
## Loading module: bugs: ok
## . . Reading data file data.txt
## . Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 1467
##    Unobserved stochastic nodes: 2863
##    Total graph size: 14895
## . Reading parameter file inits1.txt
## . Initializing model
## . Adapting 5000
## -------------------------------------------------| 5000
```

```
## ++++++++++++++++++++++++++++++++++++++++++++++++++ 100%
## Adaptation successful
## . Updating 30000
## --------------------------------------------------| 30000
## ************************************************** 100%
## . . Failed to set trace monitor for return_mu_N
## Variable return_mu_N not found
## . Updating 50000
## --------------------------------------------------| 50000
## ************************************************** 100%
## . . . . Updating 0
## . Deleting model
## .
## All chains have finished
## Simulation complete.  Reading coda files...
## Coda files loaded successfully
## Note: Summary statistics were not produced as there are >50 monitored
## variables
## [To override this behaviour see ?add.summary and ?runjags.options]
## FALSEFinished running the simulation
```

```
pred_R = do.call(rbind.data.frame, as.mcmc.list(runJagsOut1, 'return_hidden_1
')) # R[i,t+1]
pred_R[,407] # 1st person's random effect
```

```
##     [1] -7.14600e-01 -1.22905e+00 -1.65027e+00 -1.04081e+00 -1.23749e+00
##     [6] -1.16104e+00 -9.65679e-01 -1.61306e+00 -1.43752e+00 -1.79153e+00
##    [11] -1.36027e+00 -2.81013e+00 -1.58220e+00 -1.10232e+00 -1.80284e-01
##    [16] -4.55128e-01 -5.02713e-01 -1.11133e+00 -3.31611e-01 -4.46360e-01
##    [21] -2.45117e-01 -8.81058e-01 -6.42776e-01 -1.11089e+00 -1.16827e+00
…
```

```
K = dim(pred_R)[1] # num of simulation
J = dim(pred_R)[2] # num of people

R6 <- colMeans(exp(pred_R))
mylamb = exp(X %*% post_beta)

pred_N1 = rep(0, J)
for (i in 1:J){
  pred_N1[i] = R6[i]*mylamb[i]
}

# Test mse, mae
idx = unique(mydata$PolicyNum) %in% data.valid$PolicyNum
mse = sum((data.valid$n - pred_N1[idx])^2 / J)
mae = sum(abs(data.valid$n - pred_N1[idx]) / J)
c(mse, mae)
```

```
## [1] 0.9619265 0.4005591
```

4> 우선 2>의 결과를 이용하여 JAGS SSM 모델의 parameter 들을 $\hat{\phi}, \sigma^2, \hat{\beta_0}, \cdots, \hat{\beta_p}$을 사후분포의 median 값으로 고정시킵니다. 이 문제에서는 $E[N_{i,t+1} | n_{i,1}, \cdots, n_{i,t}]$의 근사치를 얻기 위해 Credibility 방법을 사용합니다.

4-1> 우선 다음과 같은

mycov(t1,t2, lamb1, lamb2)

$cov(N_{t_1}, N_{t_2})$을 return하는 R-function을 작성하세요. 단, $t_1 = t_2$인 경우 $Var(N_{t_1})$을 return하게 하세요.

```
# ========= Problem4 =========
mycov <- function(t1, t2, lamb1, lamb2){
  if (t1==t2){
    var_R = post_sigsq + post_phi^2 * post_sigsq / (1 - post_phi^2)
    return(lamb1*lamb2*var_R)
  }else{
    var_R = post_phi * post_sigsq / (1 - post_phi^2)
    return(lamb1*lamb2*var_R)
  }
}
```

4-2> 이제 임의로 주어진 과거 관측치 (길이도, 관측된 해도 다를 수 있음)의 covariance matrix를 return하는 함수를 작성하세요.

covMat(t_vec, lamb_vec)

예를 들어

(n_1, n_3, n_5)가 관측되었을 경우

t_vec=c(1,3,5)

lamb_vec = (lamb1, lamb3, lamb5)로 입력값을 넣으면 다음의 matrix를

$$\begin{pmatrix} Var(n_1) & cov(n_1,n_3) & cov(n_1,n_5) \\ cov(n_1,n_3) & Var(n_3) & cov(n_3,n_5) \\ cov(n_1,n_5) & cov(n_3,n_5) & Var(n_5) \end{pmatrix}$$

return합니다.

```
covMat <- function(t_vec, lamb_vec){ #t_vec = n
  t_vec = na.omit(t_vec)
  t_vec = t_vec[ifelse(t_vec!=0,T,F)]
  t = length(t_vec)
  res = matrix(rep(0, t^2), nrow=t)
  for (i in 1:t){
    for (j in 1:t){
      res[i,j] = mycov(i, j, lamb_vec[i], lamb_vec[j])
    }
```

```
  }
  return(res)
}
```

4-3> 4-1과 4-2에서 작성된 함수를 사용하여 $E[N_{i,t+1}|n_{i,1},\cdots,n_{i,t}]$의 Credibility 추정치를 얻으세요. 이때 이 추정치를 $\widehat{E_2}[N_{i,t+1}|n_{i,1},\cdots,n_{i,t}]$로 표기합니다. 이를 사용하여 실제 data.train과 data.valid를 사용한 Test MSE와 Test MAE 를 계산하세요.

```
rownames(data.valid) = NULL
X_new <- merge(final_data[,c(1,2)], data.valid[,c(1,9,10,12,13,14)], key='Pol
icyNum', all.x=TRUE)
X_new <- X_new[,-c(1,2)]
X_new <- cbind(rep(1, dim(X_new)[1]), X_new)
myindex <- unique(mydata$PolicyNum) %in% data.valid$PolicyNum
n_new = sum(myindex)

# Lambda 에 들어갈 random effect(R) : 시간에 따라 변화
R <- matrix(rep(0, J*6), nrow=J)
for (j in 1:J){
  R[j,1] = dnorm(0, post_sigsq/(1-post_phi^2))
  for (n in 2:6){
    R[j,n] = dnorm(post_phi*R[j,n-1], post_phi*post_sigsq/(1-post_phi^2))
  }
}

# Lambda : 시간(t)에 따라 변화
lambda <- exp(R)[,1:5] * c(exp(as.matrix(X) %*% as.matrix(post_beta)))
lambda
```

```
##              [,1]       [,2]       [,3]       [,4]       [,5]
##   [1,] 2.75969428 2.93777296 2.97831419 2.98709226 2.98897105
##   [2,] 2.75969428 2.93777296 2.97831419 2.98709226 2.98897105
##   [3,] 2.75969428 2.93777296 2.97831419 2.98709226 2.98897105
##   [4,] 2.75969428 2.93777296 2.97831419 2.98709226 2.98897105
…
```

```
pred_lambda2 = exp(as.matrix(X_new) %*% as.matrix(post_beta)) * exp(R)[,6]
pred_lambda2
```

```
##              [,1]
##   [1,] 2.98937216
##   [2,] 2.98937216
##   [3,] 2.98937216
##   [4,] 2.98937216
…
```

```r
# Buhlmann premium 계산 (by credibility method)
bp <- rep(0, J)
for (j in 1:J){
  d = ifelse(N[j,]==0, NA, N[j,])
  idx = !is.na(d)
  idx_n = sum(idx)
  if (idx_n > 0){
    C <- matrix(rep(0, idx_n))
    for (t in 1:idx_n){
      var_R = post_phi * post_sigsq / (1 - post_phi^2)
      C[t] = lambda[j,t]*pred_lambda2[j]*var_R
    }
    V <- covMat(N[j,], lambda[j,])
    A <- solve(V) %*% C
    A0 <- pred_lambda2[j] - sum(A * lambda[j,][idx])
    AA <- c(A0, A)
    bp[j] <- c(1, N[j,][idx]) %*% AA
  }
}


# test mse
idx = unique(mydata$PolicyNum) %in% data.valid$PolicyNum
mse = sum((data.valid$n - bp[idx])^2 / J)
mae = sum(abs(data.valid$n - bp[idx]) / J)
c(mse, mae)
```

```
## [1] 1.061444 0.521639
```

6> JAGS Program 을 사용하여 다음 Poisson-BGAR(1) SSM 모델의 parameter 들 $(\phi, \sigma^2, \beta_0, \cdots, \beta_p)$의 사후 분포를 구하고 각 parameter들의 95% confidence를 구하세요.

$$N_t | R_t \sim Pois(\lambda_t R_t)$$

$$R_t = B_t * R_{t-1} + G_t$$

where $B_t \sim Beta(r_1\rho, r_2(1-\rho))$, $G_t \sim Gamma(r_1(1-\rho), r_2)$ and $R_0 \sim Gamma(r_1, r_2)$. We further assume that $r_1 = r_2$ so that we have $E[R_t] = 1$.

이때 N의 경우 t=1, …, 5까지만 사용합니다. t=6일 경우 Test set 으로 남겨 놓습니다.
Hint: 주어진 JAGS프로그램을 참조할수 있음. 문제 2~5의 과정을 반복하세요.

```r
# ========= Problem6 =========
## SSM : Pois-BGAR(1) ##
###### BGAR : R[t] = B*R[t-1] + G[t]
###### B[t] ~ Beta
###### G[t] ~ Gamma
```

```r
# problem1

res1 = mydata[which(mydata$PolicyNum == unique(mydata$PolicyNum)[1]),][1,]
for (i in 2:length(unique(mydata$PolicyNum))){
  sam = mydata[which(mydata$PolicyNum == unique(mydata$PolicyNum)[i]),]
  row = sam[which(sam$Year==min(sam$Year)),]
  res1 = rbind(res1, row)
}
res1 = res1[,-2] # remove Year
head(res1)
```

```
##    PolicyNum  ClaimBC  ClaimIM  ClaimPN  ClaimPO ClaimCN  ClaimCO TypeCity
## 1     120002     0.00     0.00     0.00     0.00    0.00  6275.06        0
## 6     120003     0.00 21589.93     0.00     0.00    0.00  4206.64        0
## 11    120004 13003.08     0.00 22405.56  2712.53 1032.26     0.00        0
## 16    120005     0.00  2775.65  3834.33     0.00 1722.52 38167.20        0
## 21    120008 48060.82     0.00  4235.19     0.00    0.00 41587.00        0
## 31    120010 20110.06   796.90  7369.74 32776.10  780.82  1589.80        0
##    TypeCounty TypeMisc TypeSchool TypeTown TypeVillage IsRC CoverageBC
## 1           1        0          0        0           0    1   22.71446
## 6           1        0          0        0           0    1   99.35338
## 11          1        0          0        0           0    1   25.87963
## 16          1        0          0        0           0    1   31.46258
## 21          1        0          0        0           0    1   46.05153
## 31          1        0          0        0           0    1   71.01750
##    lnDeductBC NoClaimCreditBC    yAvgBC FreqBC CoverageIM lnDeductIM
## 1    6.907755               0     0.000      0   3.430260   6.907755
## 6    8.517193               0     0.000      0   9.198051   8.517193
## 11   6.214608               0  4334.360      3   4.055599   6.214608
## 16   6.907755               0     0.000      0   4.633063   6.214608
## 21   6.214608               0 16020.273      3   2.754886   6.214608
## 31  10.126631               0  5027.515      4   4.089611   8.517193
##    NoClaimCreditIM   yAvgIM FreqIM CoveragePN NoClaimCreditPN    yAvgPN Fr
eqPN
## 1                0     0.00      0   0.724329               0    0.0000
   0
## 6                0 10794.97      2   1.192516               0    0.0000
   0
## 11               0     0.00      0   0.591822               0 5601.3900
   4
## 16               0  2775.65      1   1.099368               0  958.5825
   4
## 21               0     0.00      0   0.943535               0 1411.7300
   3
## 31               0   398.45      2   0.444427               0  921.2175
   8
##    CoveragePO NoClaimCreditPO    yAvgPO FreqPO CoverageCN NoClaimCreditCN
## 1    1.692349               0    0.0000      0   0.724329               0
## 6    4.087988               0    0.0000      0   1.192516               0
```

```
## 11   0.518618               0  904.1767      3   0.591822               0
## 16   2.900052               0    0.0000      0   1.099368               0
## 21   1.091113               0    0.0000      0   0.943535               0
## 31   4.262824               0 8194.0250      4   0.444427               0
##     yAvgCN FreqCN CoverageCO NoClaimCreditCO    yAvgCO FreqCO col.freq  co
l.Cov
## 1     0.00      0   1.831077               0  3137.530      2        2 2.5
55406
## 6     0.00      0   4.034988               0  1402.213      3        3 5.2
27504
## 11 1032.26      1   0.518618               0     0.000      0        1 1.1
10440
## 16 1722.52      1   2.900052               0 38167.200      1        2 3.9
99420
## 21    0.00      0   1.091113               0 41587.000      1        1 2.0
34648
## 31  390.41      2   4.262824               0   794.900      2        4 4.7
07251
##    col.Cov.Idx n        s
## 1            2 2  6275.06
## 6            2 3  4206.64
## 11           1 1  1032.26
## 16           2 2 39889.72
## 21           2 1 41587.00
## 31           2 4  2370.62
```

```
res2 = as.data.frame(acast(data=mydata, PolicyNum ~ Year, value.var='n', fill
=NA))
res2$PolicyNum = unique(mydata$PolicyNum)
res2 = merge(res2, data.valid[,c('PolicyNum', 'n')], by='PolicyNum', all=TRU
E)
colnames(res2) = c('PolicyNum', paste0('n_', seq(2006,2011)))
head(res2)
```

```
##   PolicyNum n_2006 n_2007 n_2008 n_2009 n_2010 n_2011
## 1    120002      2      0      4      2      0      1
## 2    120003      3      4      3      3      4      3
## 3    120004      1      2      0      4      4      1
## 4    120005      2      4      6      1      3      1
## 5    120008      1      4      3      1      1      1
## 6    120010     NA     NA     NA      4      3      3
```

```
data = merge(res2, res1, by='PolicyNum')
final_data2 = subset(data, select = c('PolicyNum', 'n_2006', 'n_2007', 'n_200
8', 'n_2009', 'n_2010', 'n_2011',
                                      'TypeCity', 'TypeCounty', 'TypeSchool',
'TypeTown', 'TypeVillage',
                                      'col.Cov.Idx'
))
```

```r
# Result
head(final_data2)
```

```
##   PolicyNum n_2006 n_2007 n_2008 n_2009 n_2010 n_2011 TypeCity TypeCounty
## 1    120002      2      0      4      2      0      1        0          1
## 2    120003      3      4      3      3      4      3        0          1
## 3    120004      1      2      0      4      4      1        0          1
## 4    120005      2      4      6      1      3      1        0          1
## 5    120008      1      4      3      1      1      1        0          1
## 6    120010     NA     NA     NA      4      3      3        0          1
##   TypeSchool TypeTown TypeVillage col.Cov.Idx
## 1          0        0           0           2
## 2          0        0           0           2
## 3          0        0           0           1
## 4          0        0           0           2
## 5          0        0           0           2
## 6          0        0           0           2
```

```r
# problem2

X <- unname(as.matrix(final_data[8:12]))
X <- cbind(matrix(rep(1, dim(X)[1])), X)
N <- unname(as.matrix(final_data[,2:6]))
TT_N <- apply(!is.na(N), 1, sum)

# T1: indicator matrix that indicates the year of non NA N
T1 <- matrix(nrow=nrow(N), ncol=5)
for(i in 1:nrow(N)){
  ind <- which(!is.na(N[i,]))
  ind <- append(ind, rep(NA, 5-length(ind)))
  T1[i,] <- ind
}

head(T1)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    1    2    3    4    5
## [3,]    1    2    3    4    5
## [4,]    1    2    3    4    5
## [5,]    1    2    3    4    5
## [6,]    4    5   NA   NA   NA
```

```r
TTT = rep(5, dim(X)[1])

# datalist
dataList=list(N=N, X=X, TTT=TTT, TT_N=TT_N, T1=T1,
              m.beta=c(-3, rep(0,dim(X)[2]-1)), # initial value
              inv_Sigma=1*diag(dim(X)[2]), rho=0.3,
```

```
              K = dim(X)[1], c0=0.001, d0=0.001 # hyperparameter
)

#initial value for the parameters
init.beta= c(-3, rep(0,dim(X)[2]-1))

# model
modelString="model {

for(i in 1:K){
    R[i,1] ~ dgamma(r1, r1)
    for(t in 2:(TTT[i]+2)){ #TTT = 5 #number of years regardless of observed
or not.
        R[i,t] = bigbeta * R[i,t-1] + bigg
    }
    return_hidden_1[i] = R[i,TTT[i]+2] # R[i,t+1]
}


## Likelihood Part

for(i in 1:K){ #K: number of people
    for(t in 1:TT_N[i]){ #TT_N[i]=3: number of total observed years = 3 / 5
                        #T1[i,]= [1,3,5,NA, NA] # indication of observed years
        mu_N[i,T1[i,t]] = exp( inprod(X[i,],beta_hat[])) * exp(R[i,T1[i,t]+
1])
        N[i,T1[i,t]] ~ dpois( mu_N[i,T1[i,t]] )
    }
}

## Prior

r1 ~ dgamma(c0,d0)
bigbeta ~ dbeta(r1*rho, r1*(1-rho))
bigg ~ dgamma(r1*(1-rho), r1)
beta_hat[1:length(m.beta)] ~ dmnorm(m.beta[], inv_Sigma)
}
"


# run jags
inits1=list(beta_hat=init.beta,.RNG.name="base::Super-Duper")
inits2=list(beta_hat=init.beta,.RNG.name="base::Wichmann-Hill")
inits3=list(beta_hat=init.beta,.RNG.name="base::Mesenne-Twister")

nChains=3; nAdapt=5000; nUpdate=30000; nSamples=30000; nthin=5
ptm.init <- proc.time()
runJagsOut = run.jags(method="parallel", model=modelString,
                      monitor=c("beta_hat", 'bigbeta', 'bigg'),
```

```
                      data=dataList, inits=list(inits1, inits2, inits3),
                      n.chains=nChains, adapt=nAdapt, burnin=nUpdate,
                      sample=ceiling(nSamples/nChains), thin=nthin,
                      summarise=TRUE, plots=TRUE)
```

```
## Calling 3 simulations using the parallel method...
## Following the progress of chain 1 (the program will wait for all chains
## to finish before continuing):
## Welcome to JAGS 4.3.0 on Sat May 22 15:52:47 2021
## JAGS is free software and comes with ABSOLUTELY NO WARRANTY
## Loading module: basemod: ok
## Loading module: bugs: ok
## . . Reading data file data.txt
## . Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 1467
##    Unobserved stochastic nodes: 413
##    Total graph size: 14934
## . Reading parameter file inits1.txt
## . Initializing model
## . Adapting 5000
## -------------------------------------------------| 5000
## +++++++++++++++++++++++++++++++++++++++++++++++++ 100%
## Adaptation successful
## . Updating 30000
## -------------------------------------------------| 30000
## ************************************************** 100%
## . . . . Updating 50000
## -------------------------------------------------| 50000
## ************************************************** 100%
## . . . . Updating 0
## . Deleting model
## .
## All chains have finished
## Simulation complete.  Reading coda files...
## Coda files loaded successfully
## Calculating summary statistics...
## Calculating the Gelman-Rubin statistic for 8 variables....
## Finished running the simulation
```

```
summary(runJagsOut)[,c(1,3)]
```

```
##                  Lower95    Upper95
## beta_hat[1] -3.15632000 -1.9474600
## beta_hat[2]  0.20171600  1.5564600
## beta_hat[3]  1.69615000  2.9513200
## beta_hat[4] -0.65836900  0.6088290
```

```
## beta_hat[5] -2.00784000 -0.3985570
## beta_hat[6] -0.76957800  0.5892100
## bigbeta      0.97778600  1.0000000
## bigg         0.00369665  0.0948494

# problem3
post_beta = summary(runJagsOut)[1:6,2]
post_bigbeta = summary(runJagsOut)[7,2]
post_bigg = summary(runJagsOut)[8,2]

# datalist
dataList=list(N=N, X=X, TTT=TTT, TT_N=TT_N, T1=T1,
              post_beta=post_beta, post_bigbeta=post_bigbeta, post_bigg=post_
bigg,
              c0=0.001, d0=0.001, K = dim(X)[1] # hyperparameter
)

# model
modelString="model {

###### Prior ######
for(i in 1:K){
    R[i,1] ~ dgamma(r1, r1)
    for(t in 2:(TTT[i]+2)){ #TTT = 5 #number of years regardless of observed
or not.
        R[i,t] = post_bigbeta * R[i,t-1] + post_bigg
    }
    return_hidden_1[i] = R[i,TTT[i]+2] # R[i,t+1]
}

###### Likelihood Part ######
for(i in 1:K){ #K: number of people
    for(t in 1:TT_N[i]){ #TT_N[i]=3: number of total observed years = 3 / 5
                         #T1[i,]= [1,3,5,NA, NA] # indication of observed years
        mu_N[i,T1[i,t]] = exp( inprod(X[i,],post_beta[])) * exp(R[i,T1[i,t]+
1])
        N[i,T1[i,t]] ~ dpois( mu_N[i,T1[i,t]] )
    }
    return_mu_N[i] = mu_N[i,T1[i,TT_N[i]]]
}

###### Prior ######
r1 ~ dgamma(c0,d0)
}
"

# run jags
inits1=list(.RNG.name="base::Super-Duper")
```

```
inits2=list(.RNG.name="base::Wichmann-Hill")
inits3=list(.RNG.name="base::Mesenne-Twister")
nChains=3; nAdapt=5000; nUpdate=30000; nSamples=30000; nthin=5
ptm.init <- proc.time()
runJagsOut1 = run.jags(method="parallel", model=modelString,
                       monitor=c("return_hidden_1", "return_mu_N", 'r1'),
                       data=dataList,
                       n.chains=nChains, adapt=nAdapt, burnin=nUpdate,
                       sample=ceiling(nSamples/nChains), thin=nthin,
                       summarise=TRUE, plots=TRUE)

## Warning: No initial values were provided - JAGS will use the same initial
values
## for all chains

## Warning: You attempted to start parallel chains without setting different
PRNG
## for each chain, which is not recommended. Different .RNG.name values have
been
## added to each set of initial values.

## Calling 3 simulations using the parallel method...
## Following the progress of chain 1 (the program will wait for all chains
## to finish before continuing):
## Welcome to JAGS 4.3.0 on Sat May 22 16:08:22 2021
## JAGS is free software and comes with ABSOLUTELY NO WARRANTY
## Loading module: basemod: ok
## Loading module: bugs: ok
## . . Reading data file data.txt
## . Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 1467
##    Unobserved stochastic nodes: 410
##    Total graph size: 14891
## . Reading parameter file inits1.txt
## . Initializing model
## . Adapting 5000
## -------------------------------------------------| 5000
## +++++++++++++++++++++++++++++++++++++++++++++++++ 100%
## Adaptation successful
## . Updating 30000
## -------------------------------------------------| 30000
## ************************************************* 100%
## . . . . Updating 50000
## -------------------------------------------------| 50000
## ************************************************* 100%
## . . . . Updating 0
```

```
## . Deleting model
## .
## All chains have finished
## Simulation complete.  Reading coda files...
## Coda files loaded successfully
## Note: Summary statistics were not produced as there are >50 monitored
## variables
## [To override this behaviour see ?add.summary and ?runjags.options]
## FALSEFinished running the simulation
```

```
pred_R = do.call(rbind.data.frame, as.mcmc.list(runJagsOut1, 'return_hidden_1
')) # R[i,t+1]
pred_R[,407] # 1st person,
```

```
list_r1 = do.call(rbind.data.frame, as.mcmc.list(runJagsOut1, 'r1'))
```

```
K = dim(pred_R)[1] # num of simulation
J = dim(pred_R)[2] # num of people
```

```
# Buhlmann Premium
pred_N1 = rep(0, J)
for (i in 1:J){
  res = exp(pred_R[,i]) * pred_mu_N[,i]
  pred_N1[i] = sum(res) / K
}
```

```
# Test mse, mae
idx = unique(mydata$PolicyNum) %in% data.valid$PolicyNum
mse = sum((data.valid$n - pred_N1[idx])^2 / J) # 크게 튀는 값들이 있어서 그런듯
mae = sum(abs(data.valid$n - pred_N1[idx]) / J)
c(mse, mae)
```

```
## [1] 1601.8971    8.1636
```

```
# problem4
rho = 0.3
mycov <- function(t1, t2, lamb1, lamb2){
  if (t1==t2){
    var_R = 1 / r1
    return(lamb1*lamb2*var_R)
  }else{
    var_R = rho *abs(t2 - t1) / r1
    return(lamb1*lamb2*var_R)
  }
}

covMat <- function(t_vec, lamb_vec){ #t_vec = n
  t_vec = na.omit(t_vec)
  t_vec = t_vec[ifelse(t_vec!=0,T,F)]
```

```
  t = length(t_vec)
  res = matrix(rep(0, t^2), nrow=t)
  for (i in 1:t){
    for (j in 1:t){
      res[i,j] = mycov(i, j, lamb_vec[i], lamb_vec[j])
    }
  }
  return(res)
}

rownames(data.valid) = NULL
X_new <- merge(final_data[,c(1,2)], data.valid[,c(1,9,10,12,13,14)], key='Pol
icyNum', all.x=TRUE)
X_new <- X_new[,-c(1,2)]
X_new <- cbind(rep(1, dim(X_new)[1]), X_new)
myindex <- unique(mydata$PolicyNum) %in% data.valid$PolicyNum
n_new = sum(myindex) # train 과 test 모두 존재하는 policynum 의 개수


# Lambda 에 들어갈 random effect
r1 <- apply(list_r1, 2, mean)[1] # 30000 simulation 의 평균
R <- matrix(rep(0, J*6), nrow=J)
for (j in 1:J){
  R[j,1] = dgamma(r1, r1)
  for (n in 2:6){
    R[j,n] = post_bigbeta * R[j,n-1] + post_bigg
  }
}

lambda <- exp(R)[,1:5] * c(exp(as.matrix(X) %*% as.matrix(post_beta))) # lamb
da : 시간에 따라 변화
pred_lambda2 = exp(as.matrix(X_new) %*% as.matrix(post_beta)) * exp(R)[,6]

# Buhlmann Premium (by Creidiblity Method)
bp <- rep(0, J)
rho <- 0.3
for (j in 1:J){
  d = ifelse(N[j,]==0, NA, N[j,]) # N ==0 이면 NA 로 변환
  idx = !is.na(d) # 값이 0 이거나 NA(존재하지 않음)인 경우
  idx_n = sum(idx) # 총 경우의 수
  if (idx_n > 0){
    C <- matrix(rep(0, idx_n))
    for (t in 1:idx_n){
      var_R = 1 / r1 * rho * (6 -t)
      C[t] = lambda[j,t]*pred_lambda2[j]*var_R
    }
```

```
    V <- covMat(N[j,], lambda[j,])
    A <- solve(V) %*% C
    A0 <- pred_lambda2[j] - sum(A * lambda[j,][idx])
    AA <- c(A0, A)
    bp[j] <- c(1, N[j,][idx]) %*% AA
  }
}
bp
```

```
##   [1]   5.6903438   4.4565122  75.8360827   4.4199639   1.4500677   7.4739
231
##   [7]  51.7551483   5.7422747   6.4894164   4.3313297   0.0000000  -3.7655
356
```

…

```
# Test mse, mae
idx = unique(mydata$PolicyNum) %in% data.valid$PolicyNum
mse = sum((data.valid$n - bp[idx])^2 / J)
mae = sum(abs(data.valid$n - bp[idx]) / J)
c(mse, mae)
```

```
## [1] 69.535307  2.457668
```