금융위험관리 기말과제

202STG01 고유정 박소연 202STG16 신승은 202STG19 이채영 202STG20 이혜린

NMC와 LSMC를 이용한 사고건수 예측

1. NMC

```
## Parameters

I=5000
tau=5
beta=c(-3.5,1,2)
phi=0.95
sig=0.5

set.seed(1234)

##################################################
# 1> NMC
##################################################
set.seed(1)

RMSE_NMC=matrix(0,nrow=50,ncol=1)
B=1:50
K=100

ptm.init <- proc.time()

for(b in B){
  # X
  X = matrix(0,nrow=I,ncol=3)
  for(i in 1:I){
    X[i,]<- cbind(1,rbinom(1,1,0.5),rbinom(1,1,0.5))
  }

  # Lambda: time-invariant
  Lamb = exp(X%*%beta)

  # R : AR(1), Y
  R = matrix(0,nrow=I,ncol=7)
  Y = matrix(0,nrow=I,ncol=6)
  for(i in 1:I){
    R[i,1] = rnorm(1, 0, sqrt(sig/(1-phi^2)))
    for(t in 2:7){
      R[i,t] = rnorm(1, phi*R[i,t-1], sqrt(sig))
    }
    Y[i,] = rpois(6,Lamb[i]*exp(R[i,c(2:7)]))
  }

  ## 1-1> Explain how you simulate y for given Y and lambda. Here, we have tau=5
  # data list
  dataList=list(I=I,X=X,Y=Y,beta=beta,sig=sig,phi=phi,tau=tau)

  # model
  modelString="model {
  for(i in 1:I){
      R[i,1] ~ dnorm(0, (1-phi^2)/sig)
      for(t in 2:(tau+2)){
          R[i,t] ~ dnorm(phi*R[i,t-1], 1/sig)
      }
      R6[i] = exp(R[i,(tau+2)]) #save R_6
```

```
    }

  for(i in 1:I){ #I: number of people
      for(t in 1:6){
          mu_N[i,t] = exp( inprod(X[i,],beta[]) + R[i,t+1] )
          Y[i,t] ~ dpois( mu_N[i,t] )
      }
    }
  }
  "
  writeLines(modelString, "model_reg.txt")

  nChains=1 # K=100
  jagsModel = jags.model(file="model_reg.txt", data=dataList, n.chains=nChains, n.adapt=1000)
  update(jagsModel, n.iter=100)
  codaSamples = coda.samples(jagsModel, variable.names=c("R6"), n.iter=100)
  exp_R6 = colMeans(codaSamples[[1]])
  prem = Lamb*exp_R6
  RMSE_NMC[b] = sqrt(mean((prem-Y[,6])^2))
  print(paste0("=== iteration : ",b," ==="))
  print(RMSE_NMC[b])
}

time_5000_NMC<- round(mean(proc.time()-ptm.init)/60,2)

# 1-2-1> Calculate the average RMSE for Scenario 2 for B iterations

avg_RMSE_NMC = mean(RMSE_NMC)
print(paste0("=== RMSE_NMC : ",round(avg_RMSE_NMC,2)," ==="))
```

RMSE = 0.42

2. Simple LSMC

```
param_beta =c(-3.5,1,2)
phi = 0.95
sig = 0.5
I=5000
# ======== Problem2 ========
###### Simple LSMC ######

###### Step1. Sampling with JAGS ######
## # of data = 5000

I = 5000
X = matrix(0, nrow=I, ncol=2)
for (i in 1:I){
  X[i,1] = rbinom(1, 1, 0.5) # independent
  X[i,2] = rbinom(1, 1, 0.5)
}
X <- cbind(rep(1, I), X)

## Scenario2
beta = c(-3.5, 1, 2)
phi = 0.95
sigsq = 0.5
lambda = exp(X %*% beta)
R = matrix(0, nrow=I, ncol=7)
Y = matrix(0, nrow=I, ncol=6)
for (i in 1:I){
  R[i,1] = rnorm(1, 0, sqrt(sigsq/(1-phi^2)))
  for (t in 1:6){
    R[i,(t+1)] = rnorm(1, phi*R[i,t], sqrt(sigsq))
  }
  Y[i,] = rpois(6,lambda[i]*exp(R[i,c(2:7)]))
}

## datalist
```

```r
dataList=list(lambda=lambda, I=I, sig=sig, phi=phi # hyperparameter
)

## model
modelString="model {

  ######## Prior ########
  for(i in 1:I){
      R[i,1] ~ dnorm(0, (1-phi^2)/sig)
      for(t in 2:7){ # R_{0}, ..., R_{6}
          R[i,t] ~ dnorm(phi*R[i,t-1], 1/sig)
          R_star[i,t] = exp(R[i,t])
      }
  }

  ######## Y ########
  for (i in 1:I){
    for (t in 1:6){
      mu_Y[i,t] = lambda[i,] * R_star[i,(t+1)]
      Y[i,t] ~ dpois(mu_Y[i,t])
    }
  }
  }
  "
writeLines(modelString, 'model_ex4.txt')

## run jags
jagsModel = jags.model(file='model_ex4.txt', data=dataList,
                       n.chains=1, n.adapt=500
)

update(jagsModel, n.iter=5000)
codasamples = coda.samples(jagsModel, # K = 100 (n.iter)
                           variable.names=c('Y', 'R_star'), n.iter=100)
data = colMeans(codasamples[[1]])
pred_R = matrix(unname(data[1:30000]), nrow=I, ncol=6)
pred_Y = matrix(unname(data[30001:length(data)]), nrow=I, ncol=6)

###### Step2. Weighted LM ######

B = 50
res2 = rep(0, B)

for (b in 1:B){
  ## Bootstrap sampling
  K = 100000
  idx = sample(seq(1,I), K, replace=T)

  X_boot = matrix(rep(0, K*3), nrow=K)
  for (i in 1:K){
    idxx = idx[i]
    X_boot[i,] = as.matrix(X[idxx,])
  }

  lambda_boot = exp(X_boot %*% beta)
  R_boot = matrix(0, nrow=K, ncol=7)
  Y_boot = matrix(0, nrow=K, ncol=6)
  for (i in 1:K){
    R_boot[i,1] = rnorm(1, 0, sqrt(sigsq/(1-phi^2)))
    for (t in 1:5){
      R_boot[i,(t+1)] = rnorm(1, phi*R_boot[i,t], sqrt(sigsq))
    }
    Y_boot[i,] = rpois(6,lambda_boot[i]*exp(R_boot[i,c(2:7)]))
  }

  wdata = matrix(0, nrow=K, ncol=6)
  for (k in 1:K){
    wdata[k,] = as.matrix(Y_boot[k,])
  }
  wdata = data.frame(wdata)
  colnames(wdata) = paste0('predY', seq(1,6))

  ## Weighted Least Squares
```

```r
  res_lm = lm(predY6~., data=wdata, weight=lambda_boot)
  AA = as.matrix(unname(res_lm$coefficients))

  premium2 = rep(0, I)
  for (i in 1:I){
    premium2[i] = lambda[i,] * t(AA) %*% (pred_Y[i,] / lambda[i,])
  }
  res2[b] = sqrt(mean((premium2 - pred_Y[,6])^2))
}

rmse2 = mean(res2)
rmse2

## [1] 4.213363
```

RMSE = 4.21

3. RNN-LSMC

```python
# Test data
Y = pd.read_csv("Y.csv")
Lamb = pd.read_csv("Lambda.csv")
Y = np.array(Y.drop('Unnamed: 0',axis=1))
Lamb = np.array(Lamb['V1'])

print("* Lambda shape:",Lamb.shape)
print("* Y shape:",Y.shape)

# Parameters
I=5000
tau=5
beta=np.array([-3.5,1,2])
phi=0.95
sig=0.5
```

```
* Lambda shape: (5000,)
* Y shape: (5000, 6)
```


3. RNN-LSMC


```python
# simulate data function
```

```
def simulate_data(Lamb,k): # 10만개
  idx = np.random.randint(low=0, high=len(Y), size=k)
  Lamb_boot = Lamb[idx]
  Y_boot = np.full((k, 6), 999)
  R_boot = np.full((k, 7), 999)
  for i in range(k):
    for t in list(range(7)):
      if t==0 :
        R_boot[i,t] = np.random.normal(0, sig/np.sqrt(1-phi**2), size=1) #= phi*R0[i] + epsilon[i,1
      else:
        R_boot[i,t] = np.random.normal(R_boot[i,t-1]*phi, sig, size=1)

    Y_boot[i,] = np.random.poisson(lam=Lamb_boot[i]*np.exp(R_boot[i,1:7]), size=6) #rpois(1, mu_n )

  Lamb_boot_6 = np.repeat(Lamb_boot,6).reshape(-1,6)
  data_sim = np.stack([Y_boot,Lamb_boot_6],axis=2)
  return data_sim

# loss function
def my_mse(y_true,y_pred): # y_true[None,1,2](numpy), y_pred[None,1](tensor)
  lamb6 = y_true[:,:,1:]
  y6 = y_true[:,:,:1]
  y6_hat = lamb6*y_pred[:,np.newaxis,:]
  mymse = tf.keras.losses.mean_squared_error(y6, y6_hat)
  return mymse
```

```
# Define LSTM model
model = Sequential([
  LSTM(10, return_sequences=True, input_shape=[None, 2]),
  LSTM(10, return_sequences=False),
  Dense(1, activation=keras.activations.exponential), # output: r6
])

# parameters
optimizer=keras.optimizers.Adam(learning_rate=0.001)
model.compile(loss=my_mse, optimizer=optimizer)

epochs = 10
batch_size = 32
```

```
# Repeat iterations
B = 50
k = 100000

MSE_RNN=[]
T1=[]
T2=[]
for b in range(B):
  start = time.time()

  # simulate data for training
  data_sim = simulate_data(Lamb, k)
```

```
  X_train, y_train = data_sim[:train_idx,:-1,:], data_sim[:train_idx,5:,:]
  X_valid, y_valid = data_sim[train_idx:,:-1,:], data_sim[train_idx:,5:,:]

  # train model
  print("========= training: iteration = {0:} =========".format(b+1))
  history = model.fit(X_train, y_train, batch_size=batch_size, epochs=epochs,
                      validation_data=(X_valid, y_valid), verbose=0)

  T1.append(time.time()-start)

  # simulate data for test
  start = time.time()

  Lamb_est = np.repeat(Lamb,6).reshape(-1,6)
  data_sim_est = np.stack([Y,Lamb_est],axis=2) # test data
  X_test, y_test = data_sim_est[:,:-1,:], data_sim_est[:,5:,:]
  prem = model.predict(X_test)*Lamb

  T2.append(time.time()-start)

  MSE_RNN.append(model.evaluate(X_test, y_test))

# 3-1-1> Calculate the average RMSE for Scenario 2 for B iterations
RMSE_RNN = np.sqrt(np.sum(MSE_RNN))/B
print("RMSE(RNN): ", round(RMSE_RNN,4))
```

RMSE 비교

NMC	LSMC	RNN-LSMC
0.42	4.21	0.48

4. 시나리오별 RMSE

< NMC >

```
I=5000
tau=5
beta=c(-3.5,1,2)
phi=0.95
sig=0.5


# Population Assumption

case_A = cumsum(rep(1000,5))
case_B = c(); case_C = c()
for(i in 1:5){
  case_B[i] = as.integer(round(i*1000/3,0))
  case_C[i] = as.integer(round((6-i)*1000/3,0))

}
case_B = cumsum(case_B)
case_C = cumsum(case_C)
```

```
make_data<- function(case,Y){
  ind_1<- 1:case[1]
  ind_2<- (case[1]+1):case[2]
  ind_3<- (case[2]+1):case[3]
  ind_4<- (case[3]+1):case[4]
  ind_5<- (case[4]+1):case[5]

  Y_4 = matrix(NA,nrow=I,ncol=6)
  Y_4[ind_1,c(5:6)] = Y[ind_1,c(5:6)]
  Y_4[ind_2,c(4:6)] = Y[ind_2,c(4:6)]
  Y_4[ind_3,c(3:6)] = Y[ind_3,c(3:6)]
  Y_4[ind_4,c(2:6)] = Y[ind_4,c(2:6)]
  Y_4[ind_5,] = Y[ind_5,]
  return(Y_4)

}
```

[SCENARIO 1]

```
#################
## 4-1) CASE A
#################

# Data

RMSE_NMC_A=matrix(0,nrow=50,ncol=1)

B=1:50


ptm.init <- proc.time()

for(b in B){
  # X
  X = matrix(0,nrow=I,ncol=3)
  for(i in 1:I){
    X[i,1]<- 1
    X[i,2]<- rbinom(1,1,0.5)
    X[i,3]<- rbinom(1,1,0.5)
  }
  # Lambda: time-invariant
  Lamb = exp(X%*%beta)

  # R : AR(1), Y
  R = matrix(0,nrow=I,ncol=7)
  Y = matrix(0,nrow=I,ncol=6)
  for(i in 1:I){
    R[i,1] = rnorm(1, 0, sqrt(sig/(1-phi^2)))
    for(t in 2:7){
      R[i,t] = rnorm(1, phi*R[i,t-1], sqrt(sig))
    }
    Y[i,] = rpois(6,Lamb[i]*exp(R[i,c(2:7)]))
  }


  Y_data = make_data(case_A,Y)
  TT_N<- apply(!is.na(Y_data), 1, sum)
  T1 <- matrix(nrow=nrow(Y_data), ncol=6)
  for(i in 1:nrow(Y_data)){
    ind <- which(!is.na(Y_data[i,]))
    ind <- append(ind, rep(NA, 6-length(ind)))
    T1[i,] <- ind
  }


  # data list

  dataList=list(Y=Y_data,TT_N=TT_N,T1=T1,I=I,sig=sig,phi=phi,X=X,beta=beta)
```

```
  modelString="model {

  ######## Prior ########

  for(i in 1:I){
      R[i,1] ~ dnorm(0, (1-phi^2)/sig)
      for(t in 2:7){ # R_{0}, ..., R_{6}
          R[i,t] ~ dnorm(phi*R[i,t-1], 1/sig)
      }
      R6[i] = exp(R[i,7])
  }


 ######## Y ########

  for(i in 1:I){ #I: number of people
      for(t in 1:(TT_N[i])){
          mu_N[i,T1[i,t]] = exp( inprod(X[i,],beta[]) + R[i,T1[i,t]+1] )
          Y[i,T1[i,t]] ~ dpois( mu_N[i,T1[i,t]] )
      }
    }
  }

  "

  writeLines(modelString, "model_reg_4.txt")

  nChains=1 # K=100
  jagsModel = jags.model(file="model_reg_4.txt", data=dataList, n.chains=nChains, n.adapt=500)
  update(jagsModel, n.iter=100)
  codaSamples = coda.samples(jagsModel, variable.names=c("R6"), n.iter=100)
  exp_R6 = colMeans(codaSamples[[1]])
  prem = Lamb*exp_R6
  RMSE_NMC_A[b] = sqrt(mean((prem-Y[,6])^2))
  print(paste0("=== iteration : ",b," ==="))
  print(RMSE_NMC_A[b])

}
```

```
time_5000_NMC_A<- round(mean(proc.time()-ptm.init)/60,2)
```

[SCENARIO 2]

```
#################
## 4-2) CASE B
#################
# Data
RMSE_NMC_B=matrix(0,nrow=50,ncol=1)
B=1:50
K=100

ptm.init <- proc.time()

for(b in B){
  # X
  X = matrix(0,nrow=I,ncol=3)
  for(i in 1:I){
    X[i,1]<- 1
    X[i,2]<- rbinom(1,1,0.5)
    X[i,3]<- rbinom(1,1,0.5)
  }

  # Lambda: time-invariant
  Lamb = exp(X%*%beta)

  # R : AR(1), Y
  R = matrix(0,nrow=I,ncol=7)
  Y = matrix(0,nrow=I,ncol=6)
  for(i in 1:I){
    R[i,1] = rnorm(1, 0, sqrt(sig/(1-phi^2)))
    for(t in 2:7){
```

```
      R[i,t] = rnorm(1, phi*R[i,t-1], sqrt(sig))
    }
    Y[i,] = rpois(6,Lamb[i]*exp(R[i,c(2:7)]))
  }

  Y_data = make_data(case_B,Y)
  TT_N<- apply(!is.na(Y_data), 1, sum)
  T1 <- matrix(nrow=nrow(Y_data), ncol=6)
  for(i in 1:nrow(Y_data)){
    ind <- which(!is.na(Y_data[i,]))
    ind <- append(ind, rep(NA, 6-length(ind)))
    T1[i,] <- ind
  }


  # data list
  dataList=list(Y=Y_data,TT_N=TT_N,T1=T1,I=I,sig=sig,phi=phi,X=X,beta=beta)

  modelString="model {

  ######## Prior ########
  for(i in 1:I){
      R[i,1] ~ dnorm(0, (1-phi^2)/sig)
      for(t in 2:7){ # R_{0}, ..., R_{6}
          R[i,t] ~ dnorm(phi*R[i,t-1], 1/sig)
      }
      R6[i] = exp(R[i,7])
  }

 ######## Y ########
  for(i in 1:I){ #I: number of people
      for(t in 1:(TT_N[i])){
          mu_N[i,T1[i,t]] = exp( inprod(X[i,],beta[]) + R[i,T1[i,t]+1] )
          Y[i,T1[i,t]] ~ dpois( mu_N[i,T1[i,t]] )
      }
    }
  }
  "
  writeLines(modelString, "model_reg_4.txt")

  nChains=1 # K=100
  jagsModel = jags.model(file="model_reg_4.txt", data=dataList, n.chains=nChains, n.adapt=500)
  update(jagsModel, n.iter=100)
  codaSamples = coda.samples(jagsModel, variable.names=c("R6"), n.iter=100)
  exp_R6 = colMeans(codaSamples[[1]])
  prem = Lamb*exp_R6
  RMSE_NMC_B[b] = sqrt(mean((prem-Y[,6])^2))
  print(paste0("=== iteration : ",b," ==="))
  print(RMSE_NMC_B[b])
}
# 1-2-1> Calculate the average RMSE for Scenario 2 for B iterations

avg_RMSE_NMC_B = mean(RMSE_NMC_B)
print(paste0("=== RMSE_NMC_B : ",round(avg_RMSE_NMC_B,2)," ==="))

## [1] "=== RMSE_NMC_B : 0.61 ==="
```

[SCENARIO 3]

```
#################
## 4-3) CASE C
#################
# Data
RMSE_NMC_C=matrix(0,nrow=50,ncol=1)
B=1:50
K=100

ptm.init <- proc.time()

for(b in B){
  # X
```

```r
  X = matrix(0,nrow=I,ncol=3)
  for(i in 1:I){
    X[i,1]<- 1
    X[i,2]<- rbinom(1,1,0.5)
    X[i,3]<- rbinom(1,1,0.5)
  }

  # Lambda: time-invariant
  Lamb = exp(X%*%beta)

  # R : AR(1), Y
  R = matrix(0,nrow=I,ncol=7)
  Y = matrix(0,nrow=I,ncol=6)
  for(i in 1:I){
    R[i,1] = rnorm(1, 0, sqrt(sig/(1-phi^2)))
    for(t in 2:7){
      R[i,t] = rnorm(1, phi*R[i,t-1], sqrt(sig))
    }
    Y[i,] = rpois(6,Lamb[i]*exp(R[i,c(2:7)]))
  }

  Y_data = make_data(case_C,Y)
  TT_N<- apply(!is.na(Y_data), 1, sum)
  T1 <- matrix(nrow=nrow(Y_data), ncol=6)
  for(i in 1:nrow(Y_data)){
    ind <- which(!is.na(Y_data[i,]))
    ind <- append(ind, rep(NA, 6-length(ind)))
    T1[i,] <- ind
  }


  # data list
  dataList=list(Y=Y_data,TT_N=TT_N,T1=T1,I=I,sig=sig,phi=phi,X=X,beta=beta)

  modelString="model {

  ######## Prior ########
  for(i in 1:I){
      R[i,1] ~ dnorm(0, (1-phi^2)/sig)
      for(t in 2:7){ # R_{0}, ..., R_{6}
          R[i,t] ~ dnorm(phi*R[i,t-1], 1/sig)
      }
      R6[i] = exp(R[i,7])
  }

 ######## Y ########
  for(i in 1:I){ #I: number of people
      for(t in 1:(TT_N[i])){
          mu_N[i,T1[i,t]] = exp( inprod(X[i,],beta[]) + R[i,T1[i,t]+1] )
          Y[i,T1[i,t]] ~ dpois( mu_N[i,T1[i,t]] )
      }
    }
  }
  "
  writeLines(modelString, "model_reg_4.txt")

  nChains=1 # K=100
  jagsModel = jags.model(file="model_reg_4.txt", data=dataList, n.chains=nChains, n.adapt=500)
  update(jagsModel, n.iter=100)
  codaSamples = coda.samples(jagsModel, variable.names=c("R6"), n.iter=100)
  exp_R6 = colMeans(codaSamples[[1]])
  prem = Lamb*exp_R6
  RMSE_NMC_C[b] = sqrt(mean((prem-Y[,6])^2))
  print(paste0("=== iteration : ",b," ==="))
  print(RMSE_NMC_C[b])
}

time_5000_NMC_C<- round(mean(proc.time()-ptm.init)/60,2)

# 1-2-1> Calculate the average RMSE for Scenario 2 for B iterations

avg_RMSE_NMC_C = mean(RMSE_NMC_C)
print(paste0("=== RMSE_NMC_C : ",round(avg_RMSE_NMC_C,2)," ==="))
```

```
## [1] "=== RMSE_NMC_C : 0.61 ==="
```

<RNN-LSMC>

Population assumption A

4-1. 데이터 생성

```python
np.random.seed(1)

start41A = time.time()

# X
nYear = 6
I = 5000
param_beta = np.array([-3.5, 1.0, 2.0])

para = pd.Series({'beta': param_beta, 'phi': 0.95, 'sig': 0.5}) # SC1

X = np.stack(( np.repeat(1,I),
                  np.random.binomial(n=1, p=0.5, size=I),
                  np.random.binomial(n=1, p=0.5, size=I)),
                 axis=1)

# parameters
sig = para['sig']
phi = para['phi']
lamb = np.exp(np.matmul(X, para['beta']))
Lamb = lamb[:, np.newaxis]
Lamb = np.concatenate([Lamb, Lamb, Lamb, Lamb, Lamb, Lamb], axis=1)

fakeY = np.ones((I, 5))

ind = np.array(np.nonzero(fakeY[0])).flatten()
ind = np.hstack([ind, [5]])
ind = np.hstack([ind, np.array([np.nan]*sum(fakeY[0]==0))])
T1 = ind

for i in range(1,I):
    ind = np.array(np.nonzero(fakeY[i])).flatten()
    ind = np.hstack([ind, [5]])
    ind = np.hstack([ind, np.array([np.nan]*sum(fakeY[i]==0))])
    T1 = np.vstack([T1,ind])

TT_Y = np.sum(~np.isnan(T1), axis=1)

Y = np.zeros((I, nYear))
pred_R = np.zeros((I, nYear+1))
```

```python
for i in range(I):
    for t in range(nYear):
        if t==0:
            pred_R[i,t] = np.random.normal(0, sig/np.sqrt(1-phi**2), size=1) # = phi*R0[i] + epsilon[i, 1]
        else:
            pred_R[i,t] = np.random.normal(pred_R[i,t-1] * phi, sig, size=1)

    for t in range(int(TT_Y[i])):
        Y[i,int(T1[i,t])] = np.random.poisson(lam = Lamb[i,int(T1[i,t])] * np.exp(pred_R[i,int(T1[i,t])]), size=1) # rpois(1, mu_n)

Y_sim = Y
lamb_sim = Lamb
data_sim = np.stack((Y_sim, lamb_sim), axis=2)

end41A = time.time()
TIME41A = end41A - start41A
print(TIME41A)
print(data_sim.shape)
```

```
2.2569656372070312
(5000, 6, 2)
```

4-2. Simulation ¶

```
np.random.seed(1)

start42B = time.time()
B = 50 # 시뮬레이션 횟수
RMSE42A = []

for b in range(B):

    # Data Resampling
    K = 100000
    seq = range(0, I)
    idx = np.random.choice(seq, size=K, replace=True)
    X_boot = []; lamb_boot = []; Y_boot = []
    for i in idx:
        X_boot.append(X[i,])
        lamb_boot.append(Lamb[i,])
        Y_boot.append(Y[i,])
    X_boot = np.array(X_boot); lamb_boot = np.array(lamb_boot); Y_boot = np.array(Y_boot)

    data_boot = np.stack((Y_boot, lamb_boot), axis=2)

    # Data split
    N_train = int(data_boot.shape[0] * 0.7)
    N_valid = int(data_boot.shape[0] * 0.9)
    N_test = int(data_boot.shape[0])

    X_train = data_boot[:N_train,:-1,:] # Y_{1} ~ Y_{5}
    X_valid = data_boot[N_train:N_valid,:-1,:]
    X_test = data_boot[N_valid:,:-1,:]

    y_train = data_boot[:N_train,-1:,:] # Y_{6}
    y_valid = data_boot[N_train:N_valid,-1:,:]
    y_test = data_boot[N_valid:,-1:,:]

    model_many_to_one = keras.models.Sequential([
        tf.keras.layers.Masking(mask_value=0.0, input_shape=[None,2]),
        keras.layers.LSTM(10, return_sequences=True),
        keras.layers.LSTM(10, return_sequences=True),
        keras.layers.LSTM(10, return_sequences=True),
        keras.layers.LSTM(10),
        keras.layers.Dense(1, activation='exponential'),
      # complete here
    ])

    def my_mse(y_true, y_pred): # y_pred : [None, 1] , y_true : [None, 1, 2]
        y6 = y_true[:,:,:1]; lamb6 = y_true[:,:,1:]
        y6_hat = lamb6 * y_pred[:,np.newaxis,:]
        mymse = tf.keras.losses.mean_squared_error(y6, y6_hat)
        mymse = tf.math.sqrt(mymse)
        return mymse

    optimizer=keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999)
    model_many_to_one.compile(loss=my_mse, optimizer=optimizer)

    epochs = 10
    batch_size=32
    print('{}th Training is Running...'.format(b+1))
    history = model_many_to_one.fit(X_train, y_train, batch_size=batch_size, epochs=epochs,  validation_data=(X_valid, y_valid), verbose

    rmse = model_many_to_one.evaluate(X_test, y_test) # num of data = 10000
    RMSE42B.append(rmse)

end42A = time.time()
TIME42A = end42A - start42A
print('Runnint Time : {}'.format(round(TIME42A, 5)))
print('Average RMSE : {}'.format(np.array(RMSE42A)[~np.isnan(RMSE42A)].mean()))
```

```
Runnint Time : 38796.38544
Average RMSE : 0.528565430027597
```

Population assumption B

4-1. 데이터 생성

```
np.random.seed(1)

start41B = time.time()

# X
nYear = 6
I = 5000
param_beta = np.array([-3.5, 1.0, 2.0])

para = pd.Series({'beta': param_beta, 'phi': 0.95, 'sig': 0.5}) # SC1

X = np.stack(( np.repeat(1,I),
                np.random.binomial(n=1, p=0.5, size=I),
                np.random.binomial(n=1, p=0.5, size=I)),
               axis=1)

# parameters
sig = para['sig']
phi = para['phi']
lamb = np.exp(np.matmul(X, para['beta']))
Lamb = lamb[:, np.newaxis]
Lamb = np.concatenate([Lamb, Lamb, Lamb, Lamb, Lamb, Lamb], axis=1)

# fake Y
def cal_k(t):
    return round(t * 1000 / 3)

def make_Y(k):
    one_array = np.concatenate([np.array([0.] * (5-k)), np.array([1.] * k)]).reshape(1,-1)
    res = one_array
    for i in range(int(cal_k(k))-1):
        res = np.vstack((res, one_array))
    return res

fakeY = np.vstack((make_Y(1), make_Y(2), make_Y(3), make_Y(4), make_Y(5)))

ind = np.array(np.nonzero(fakeY[0])).flatten()
ind = np.hstack([ind, [5]])
ind = np.hstack([ind, np.array([np.nan]*sum(fakeY[0]==0))])
T1 = ind
```

```
for i in range(1,I):
    ind = np.array(np.nonzero(fakeY[i])).flatten()
    ind = np.hstack([ind, [5]])
    ind = np.hstack([ind, np.array([np.nan]*sum(fakeY[i]==0))])
    T1 = np.vstack([T1,ind])

TT_Y = np.sum(~np.isnan(T1), axis=1)

Y = np.zeros((I, nYear))
pred_R = np.zeros((I, nYear+1))

for i in range(I):
    for t in range(nYear):
        if t==0:
            pred_R[i,t] = np.random.normal(0, sig/np.sqrt(1-phi**2), size=1) # = phi*R0[i] + epsilon[i, 1]
        else:
            pred_R[i,t] = np.random.normal(pred_R[i,t-1] * phi, sig, size=1)

    for t in range(int(TT_Y[i])):
        Y[i,int(T1[i,t])] = np.random.poisson(lam = Lamb[i,int(T1[i,t])] * np.exp(pred_R[i,int(T1[i,t])]), size=1) # rpois(1, mu_n)

Y_sim = Y
lamb_sim = Lamb
data_sim = np.stack((Y_sim, lamb_sim), axis=2)

end41B = time.time()
TIME41B = end41B - start41B
print(TIME41B)
print(data_sim.shape)
```

```
1.8231260776519775
(5000, 6, 2)
```

4-2. Simulation (Population assumption B)

```
np.random.seed(1)

start42B = time.time()
B = 50 # 시뮬레이션 횟수
RMSE42B = []

for b in range(B):

    # Data Resampling
    K = 100000
    seq = range(0, I)
    idx = np.random.choice(seq, size=K, replace=True)
    X_boot = []; lamb_boot = []; Y_boot = []
    for i in idx:
        X_boot.append(X[i,])
        lamb_boot.append(Lamb[i,])
        Y_boot.append(Y[i,])
    X_boot = np.array(X_boot); lamb_boot = np.array(lamb_boot); Y_boot = np.array(Y_boot)

    data_boot = np.stack((Y_boot, lamb_boot), axis=2)

    # Data split
    N_train = int(data_boot.shape[0] * 0.7)
    N_valid = int(data_boot.shape[0] * 0.9)
    N_test = int(data_boot.shape[0])

    X_train = data_boot[:N_train,:-1,:] # Y_{1} ~ Y_{5}
    X_valid = data_boot[N_train:N_valid,:-1,:]
    X_test = data_boot[N_valid:,:-1,:]

    y_train = data_boot[:N_train,-1:,:] # Y_{6}
    y_valid = data_boot[N_train:N_valid,-1:,:]
    y_test = data_boot[N_valid:,-1:,:]

    model_many_to_one = keras.models.Sequential([
        tf.keras.layers.Masking(mask_value=0.0, input_shape=[None,2]),
        keras.layers.LSTM(10, return_sequences=True),
        keras.layers.LSTM(10, return_sequences=True),
        keras.layers.LSTM(10, return_sequences=True),
        keras.layers.LSTM(10),
        keras.layers.Dense(1, activation='exponential'),
      # complete here
    ])

    def my_mse(y_true, y_pred): # y_pred : [None, 1] , y_true : [None, 1, 2]
        y6 = y_true[:,:,:1]; lamb6 = y_true[:,:,1:]
        y6_hat = lamb6 * y_pred[:,np.newaxis,:]
        mymse = tf.keras.losses.mean_squared_error(y6, y6_hat)
        mymse = tf.math.sqrt(mymse)
        return mymse

    optimizer=keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999)
    model_many_to_one.compile(loss=my_mse, optimizer=optimizer)

    epochs = 10
    batch_size=32
    print('{}th Training is Running...'.format(b+1))
    history = model_many_to_one.fit(X_train, y_train, batch_size=batch_size, epochs=epochs,  validation_data=(X_valid, y_valid), verbose

    rmse = model_many_to_one.evaluate(X_test, y_test) # num of data = 10000
    RMSE42B.append(rmse)

end42B = time.time()
TIME42B = end42B - start42B
print('Runnint Time : {}'.format(round(TIME42B, 5)))
print('Average RMSE : {}'.format(np.mean(RMSE42B)))
```

```
Runnint Time : 13672.34411
Average RMSE : 0.5005080145597458
```

Population assumption C

4-1. 데이터 생성

```
np.random.seed(1)

start41C = time.time()

# X
nYear = 6
I = 5000
param_beta = np.array([-3.5, 1.0, 2.0])

para = pd.Series({'beta': param_beta, 'phi': 0.95, 'sig': 0.5}) # SC1

X = np.stack(( np.repeat(1,I),
                  np.random.binomial(n=1, p=0.5, size=I),
                  np.random.binomial(n=1, p=0.5, size=I)),
                 axis=1)

# parameters
sig = para['sig']
phi = para['phi']
lamb = np.exp(np.matmul(X, para['beta']))
Lamb = lamb[:, np.newaxis]
Lamb = np.concatenate([Lamb, Lamb, Lamb, Lamb, Lamb, Lamb], axis=1)

# fake Y
def cal_k(t):
    return round((6 - t) * 1000 / 3)

def make_Y(k):
    one_array = np.concatenate([np.array([0.] * (5-k)), np.array([1.] * k)]).reshape(1,-1)
    res = one_array
    for i in range(int(cal_k(k))-1):
        res = np.vstack((res, one_array))
    return res

fakeY = np.vstack((make_Y(1), make_Y(2), make_Y(3), make_Y(4), make_Y(5)))

ind = np.array(np.nonzero(fakeY[0])).flatten()
ind = np.hstack([ind, [5]])
ind = np.hstack([ind, np.array([np.nan]*sum(fakeY[0]==0))])
T1 = ind

for i in range(1,I):
    ind = np.array(np.nonzero(fakeY[i])).flatten()
    ind = np.hstack([ind, [5]])
    ind = np.hstack([ind, np.array([np.nan]*sum(fakeY[i]==0))])
    T1 = np.vstack([T1,ind])

TT_Y = np.sum(~np.isnan(T1), axis=1)
```

```
Y = np.zeros((I, nYear))
pred_R = np.zeros((I, nYear+1))

for i in range(I):
    for t in range(nYear):
        if t==0:
            pred_R[i,t] = np.random.normal(0, sig/np.sqrt(1-phi**2), size=1) # = phi*R0[i] + epsilon[i, 1]
        else:
            pred_R[i,t] = np.random.normal(pred_R[i,t-1] * phi, sig, size=1)

    for t in range(int(TT_Y[i])):
        Y[i,int(T1[i,t])] = np.random.poisson(lam = Lamb[i,int(T1[i,t])] * np.exp(pred_R[i,int(T1[i,t])]), size=1) # rpois(1, mu_n)

Y_sim = Y
lamb_sim = Lamb
data_sim = np.stack((Y_sim, lamb_sim), axis=2)

end41C = time.time()
TIME41C = end41C - start41C
print(TIME41C)
print(data_sim.shape)
```

```
3.807821035385132
(5000, 6, 2)
```

4-2. Simulation

```
np.random.seed(1)

start42C = time.time()
B = 50 # 시뮬레이션 횟수
RMSE42C = []

for b in range(B):

    # Data Resampling
    K = 100000
    seq = range(0, l)
    idx = np.random.choice(seq, size=K, replace=True)
    X_boot = []; lamb_boot = []; Y_boot = []
    for i in idx:
        X_boot.append(X[i,])
        lamb_boot.append(Lamb[i,])
        Y_boot.append(Y[i,])
    X_boot = np.array(X_boot); lamb_boot = np.array(lamb_boot); Y_boot = np.array(Y_boot)

    data_boot = np.stack((Y_boot, lamb_boot), axis=2)
```

```
    # Data split
    N_train = int(data_boot.shape[0] * 0.7)
    N_valid = int(data_boot.shape[0] * 0.9)
    N_test = int(data_boot.shape[0])

    X_train = data_boot[:N_train,:-1,:] # Y_{1} ~ Y_{5}
    X_valid = data_boot[N_train:N_valid,:-1,:]
    X_test = data_boot[N_valid:,:-1,:]

    y_train = data_boot[:N_train,-1:,:] # Y_{6}
    y_valid = data_boot[N_train:N_valid,-1:,:]
    y_test = data_boot[N_valid:,-1:,:]

    model_many_to_one = keras.models.Sequential([
        tf.keras.layers.Masking(mask_value=0.0, input_shape=[None,2]),
        keras.layers.LSTM(10, return_sequences=True),
        keras.layers.LSTM(10, return_sequences=True),
        keras.layers.LSTM(10, return_sequences=True),
        keras.layers.LSTM(10),
        keras.layers.Dense(1, activation='exponential'),
      # complete here
    ])

    def my_mse(y_true, y_pred): # y_pred : [None, 1] , y_true : [None, 1, 2]
        y6 = y_true[:,:,:1]; lamb6 = y_true[:,:,1:]
        y6_hat = lamb6 * y_pred[:,np.newaxis,:]
        mymse = tf.keras.losses.mean_squared_error(y6, y6_hat)
        mymse = tf.math.sqrt(mymse)
        return mymse

    optimizer=keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999)
    model_many_to_one.compile(loss=my_mse, optimizer=optimizer)

    epochs = 10
    batch_size=32
    print('{}th Training is Running...'.format(b+1))
    history = model_many_to_one.fit(X_train, y_train, batch_size=batch_size, epochs=epochs,  validation_data=(X_valid, y_valid), verbose

    rmse = model_many_to_one.evaluate(X_test, y_test) # num of data = 10000
    RMSE42C.append(rmse)

end42C = time.time()
TIME42C = end42C - start42C
print('Runnint Time : {}'.format(round(TIME42C, 5)))
print('Average RMSE : {}'.format(np.mean(RMSE42C)))
```

Scenario	NMC	RNN-LSMC
1	0.60	0.53
2	0.61	0.50
3	0.61	0.58