Importance Sampling for Particle filters

Kwaku Peprah Adjei

Introduction

This document is a summary of the work done on updating a previously run occupancy model with new data. This is done by saving the weights from the previous time periods and using them as updates for the current year.

The filtering, predicting and updating stages are defined as follows:

In all cases, I simulate a data from this model with $N = 50$ years and fit the `auxilliary particle filter` under three scenarios:

- Baseline: When we have all the data to the current time point and fit the `auxilliary particle filter` to the data up to time N.
- Reduced: In this case, we have data up to time $N = 40$. We fit the `auxilliary particle filter` to the data up to time $N = 50$, with the last 10 time points having the observed data as NAs.
- Updated: In this scenario, we update the reduced model with the new 10 time points data. We use the weights from the reduced model to update the model for the last 10 time points.

Packages needed for this document

```
suppressMessages(library(nimble))
suppressMessages(library(myphdthesis)) #install_github("Peprah94/myphdthesis") to install
library(kableExtra)
library(ggplot2)
library(ggmcmc)
#library(nimbleSMC)
set.seed(1994)
```

Case 1: No top-level parameters

This section explores the performance of the `particleFilterUpdate` for models with no top level parameters. That is, aside the latent variables and the observed data, all the nodes are deterministic.

Model

$$\begin{aligned} x_0 &= 0 \\ x_t &\sim N(0.8x_{t-1}, 1) \\ y_t &\sim N(x_t, 0.5^2) \end{aligned} \tag{1}$$

Let us write some code to try out:

```
exampleCode <- nimbleCode({
  x0 ~ dnorm(0, var = 1)
  x[1] ~ dnorm(.8 * x0, var = 1)
  y[1] ~ dnorm(x[1], var = .5)
  for(t in 2:N){
    x[t] ~ dnorm(.8 * x[t-1], var = 1)
    y[t] ~ dnorm(x[t], var = .5)
  }
})
```

```
N = 50
x0 <- 0
xObs <- yObs <- numeric(N)
xObs[1] <- rnorm(1, .8*x0, sd = 1)
yObs[1] <- rnorm(1, xObs[1], sd = sqrt(0.5))
for(t in 2: N){
xObs[t] <- rnorm(1, 0.8 * xObs[t-1], sd = 1)
yObs[t] <- rnorm(1,xObs[t], sd = sqrt(0.5) )
}
```

a. We fit a `auxilliary particle filter model` assuming we have all the data available.

```
model <- nimbleModel(code = exampleCode, data = list(y = yObs), constants = list(N=N),
                     inits = list(x0 = 0, x = xObs))
baselineModel <- buildAuxiliaryFilter(model, 'x',
                                      control = list(saveAll = TRUE, lookahead = "mean", smoothi
Cmodel <- compileNimble(model)
```

```
cbaselineModel <- compileNimble(baselineModel, project = model)

logLik <- cbaselineModel$run(m = 10000)
ESS <- cbaselineModel$returnESS()

auxWts <- as.matrix(cbaselineModel$mvWSamples, "wts")
baselineWeights <- as.matrix(cbaselineModel$mvWSamples, "wts")
baselineUnweightedSamples <- as.matrix(cbaselineModel$mvWSamples, "x")
baselineWeightedSamples <- as.matrix(cbaselineModel$mvEWSamples, "x")

plotDataBaseline <- data.frame(t = seq(1,N, 1),
                        true = xObs,
                        unwtsXbaseline = apply(baselineUnweightedSamples,2,median),
                        wtsXBaseline = colMeans(baselineWeightedSamples)
                        #unwtsX = baselineUnweightedSamples[1,],
                        #wtsX = baselineWeightedSamples[1,]
                        )
```

We plot the results:

b. Case 2: Have new data for 10 years

```
reducedModel <- model$newModel(replicate = TRUE)

#set the years without data as NA
reducedModel$y[41:50] <- NA
reducedModel$x[41:50] <- NA
compileNimble(reducedModel)
```

```
Derived CmodelBaseClass created by buildModelInterface for model exampleCod_MID_1
```

```
reducedAuxModel <- buildAuxiliaryFilter(reducedModel, 'x',
                                control = list(saveAll = TRUE, lookahead = "mean", smoothi

cReducedModel <- compileNimble(reducedAuxModel, project = reducedModel)

logLik <- cReducedModel$run(m = 10000)
```

```
warning: value of data node y[41]: value is NA or NaN.
```

```
warning: logProb of data node y[41]: logProb is NA or NaN.
warning: value of data node y[42]: value is NA or NaN.
warning: logProb of data node y[42]: logProb is NA or NaN.
warning: value of data node y[43]: value is NA or NaN.
warning: logProb of data node y[43]: logProb is NA or NaN.
warning: value of data node y[44]: value is NA or NaN.
warning: logProb of data node y[44]: logProb is NA or NaN.
warning: value of data node y[45]: value is NA or NaN.
warning: logProb of data node y[45]: logProb is NA or NaN.
warning: value of data node y[46]: value is NA or NaN.
warning: logProb of data node y[46]: logProb is NA or NaN.
warning: value of data node y[47]: value is NA or NaN.
warning: logProb of data node y[47]: logProb is NA or NaN.
warning: value of data node y[48]: value is NA or NaN.
warning: logProb of data node y[48]: logProb is NA or NaN.
warning: value of data node y[49]: value is NA or NaN.
warning: logProb of data node y[49]: logProb is NA or NaN.
warning: value of data node y[50]: value is NA or NaN.
warning: logProb of data node y[50]: logProb is NA or NaN.
```

```
ESS <- cReducedModel$returnESS()

auxWts <- as.matrix(cReducedModel$mvWSamples, "wts")
reducedWeights <- as.matrix(cReducedModel$mvWSamples, "wts")
reducedUnweightedSamples <- as.matrix(cReducedModel$mvWSamples, "x")
reducedWeightedSamples <- as.matrix(cReducedModel$mvEWSamples, "x")

plotDataReduced <- data.frame(t = seq(1,N, 1),
                        true = xObs,
                        unwtsXReduced = apply(reducedUnweightedSamples,2,median),
                        wtsXReduced = colMeans(reducedWeightedSamples)
                        #unwtsX = baselineUnweightedSamples[1,],
                        #wtsX = baselineWeightedSamples[1,]
                        )
```

We plot the results:

c. Updating with new data

```
updatedModel <- model$newModel(replicate = TRUE)
```

```
#set the years without data as NA
updatedModel$x[41:50] <- NA
compileNimble(updatedModel)
```

```
Derived CmodelBaseClass created by buildModelInterface for model exampleCod_MID_1
```

```
updatedAuxModel <- buildAuxiliaryFilterUpdate(updatedModel, 'x',reducedWeights,
                                              reducedUnweightedSamples,reducedWeightedSamples,
                                    control = list(saveAll = TRUE, lookahead = "mean", smoothi

cUpdatedModel <- compileNimble(updatedAuxModel, project = updatedModel)

logLik <- cUpdatedModel$run(m = 10000)
ESS <- cUpdatedModel$returnESS()

auxWts <- as.matrix(cUpdatedModel$mvWSamples, "wts")
updatedWeights <- as.matrix(cUpdatedModel$mvWSamples, "wts")
updatedUnweightedSamples <- as.matrix(cUpdatedModel$mvWSamples, "x")
updatedWeightedSamples <- as.matrix(cUpdatedModel$mvEWSamples, "x")

plotDataUpdated <- data.frame(t = seq(1,N, 1),
                          true = xObs,
                          unwtsXUpdated = apply(updatedUnweightedSamples,2,median),
                          wtsXUpdated = colMeans(updatedWeightedSamples)
                          #unwtsX = baselineUnweightedSamples[1,],
                          #wtsX = baselineWeightedSamples[1,]
                          )
```

We plot the results:

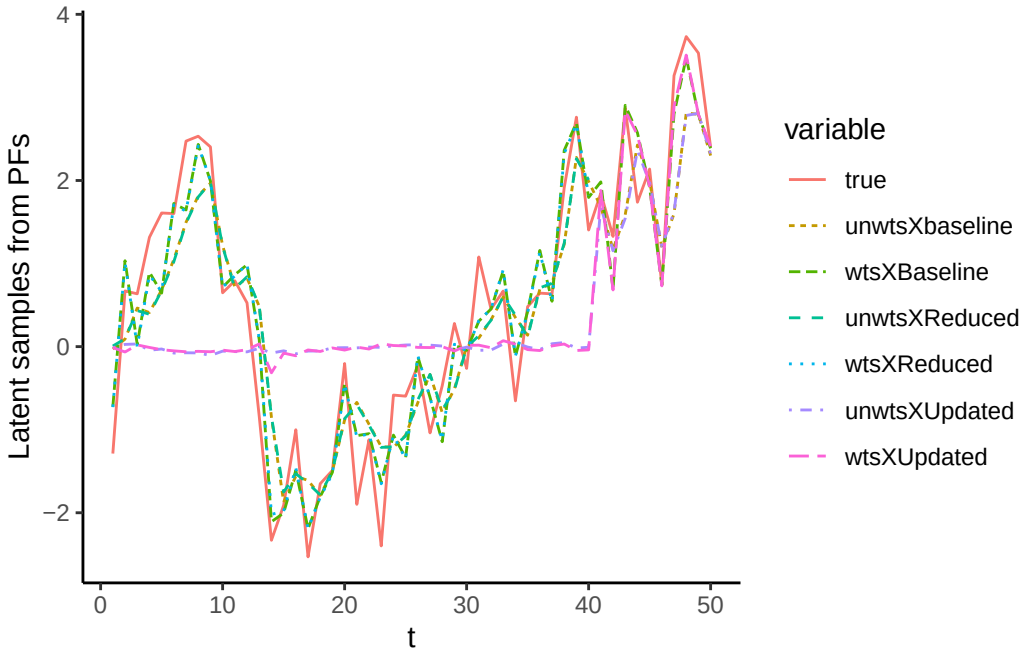
```
dfDataBaseline <- plotDataBaseline%>%
  reshape2::melt(., id.vars = "t")

dfDataReduced <- plotDataReduced%>%
  reshape2::melt(., id.vars = "t")

dfDataUpdated <- plotDataUpdated%>%
  reshape2::melt(., id.vars = "t")
```

```
plotData <- rbind(dfDataBaseline,
                  dfDataReduced ,
                  dfDataUpdated)

plotData%>%
ggplot()+
  geom_line(mapping = aes(x = t, y = value, linetype = variable, col = variable))+
  ylab("Latent samples from PFs")+
  theme_classic()
```



Case 2: MCMC for top level nodes

Model

$$
\begin{aligned}
x_1 &\sim N(\mu_0, 1) \\
x_t &\sim N(a x_{t-1} + b, 1) \\
y_t &\sim N(c * x_t, 1)
\end{aligned} \tag{2}
$$

```r
# simulate some data
sim2 <- function(a, b, t, sigPN, sigOE, df){
  x <- y <- numeric(t)
  x[1] <- rnorm(1, b/(1-a), sigPN/(sqrt(1- a^2)) )
#https://stats.stackexchange.com/questions/567944/how-can-i-sample-from-a-shifted-and-scal
  y[1] <- x[1] + (sigOE * (sqrt(df -2)/df) * rt(1, df))

for(k in 2:t){
  x[k] <- rnorm(1, a*x[k -1] + b, sigPN)
  y[k] <- x[k-1] + (sigOE * (sqrt(df -2)/df) * rt(1, df))
}
  return(list(x=x, y=y))
}

simData <- sim2(a = 0.5,
                b = 3,
                t = 50,
                sigPN = 1,
                sigOE = 1,
                df = 5)

#Set the number of iteration
parNum <- 5000
```

a. Fitting the baseline model

```r
stateSpaceCode <- nimbleCode({
  a ~ dunif(-0.9999, 0.9999)
  b ~ dnorm(0, sd = 1000)
  sigPN ~ dunif(1e-04, 1)
  sigOE ~ dunif(1e-04, 1)
  x[1] ~ dnorm(b/(1 - a), sd = sigPN/sqrt((1-a*a)))
  y[1] ~ dt(mu = x[1], sigma = sigOE, df = 5)
  for (i in 2:t) {
    x[i] ~ dnorm(a * x[i - 1] + b, sd = sigPN)
    y[i] ~ dt(mu = x[i], sigma = sigOE, df = 5)
  }
})


data <- list(
```

```
  y = simData$y
)
constants <- list(
  t = 10
)
inits <- list(
  a = 0,
  b = .5,
  sigPN = .1,
  sigOE = .05
)

## build the model
stateSpaceModel <- nimbleModel(stateSpaceCode,
                               data = data,
                               constants = constants,
                               inits = inits,
                               check = FALSE)
## build bootstrap filter and compile model and filter
AuxFilter <- buildAuxiliaryFilter(stateSpaceModel, nodes = 'x',
                                        control = list(saveAll = TRUE, lookahead = 'mean'))
compiledList <- compileNimble(stateSpaceModel, AuxFilter)
## run compiled filter with 10,000 particles.
## note that the bootstrap filter returns an estimate of the log-likelihood of the model.
compiledList$AuxFilter$run(10000)
```

```
[1] -220.6895
```

```
plotDataBaseline <- data.frame(t = seq(1,N, 1),
                        true = xObs,
                        unwtsXbaseline = apply(baselineUnweightedSamples,2,median),
                        wtsXBaseline = colMeans(baselineWeightedSamples)
                        #unwtsX = baselineUnweightedSamples[1,],
                        #wtsX = baselineWeightedSamples[1,]
                        )
```

To do for next week

In the next week, I will look at the MCMC with particle filter. I have editted the code for sampling, but there seems to be some bugs. That will be my work for next week.