Testing the model with simulated data

W. christopher Carleton

Simulating data

The aim of this document is to demonstrate that the model described in the associated paper works as expected with simulated data.

To begin, we need to simulate some data. For this purpose, we will use one of the temperature records analyzed in the associated paper as the key driver of variation in the simulated count data set against the background autocorrelated process. There are several main variables involved. These include 'y', which we will let be the count (of conflicts); 'x', which we will use as the temperature variable; and [b1, b2, b3], the regression coefficients relating temperature to count.

Load the covariate data and set the number of observations:

```
df <- read.csv(file ="../Data/EuroClimCon.csv", head = T)
t1 <- 1005
t2 <- 1980
df <- subset(df, Year >= t1 & Year <= t2)

x <- scale(df$T_Buntgen2021, scale = T)[,1]
n <- length(x)
```

Set the necessary regression coefficients including the thresholds, denoted a1 (upper) and a2 (lower).

```
b1 <- 0
b2 <- 1.25
b3 <- -1.25

a1 <- quantile(x,probs=0.95)
a2 <- quantile(x,probs=0.05)
```

Set up the variables and autocorrelation process and then simulate the final count process.

```
l <- c()
l0 <- 1
rho = 0.9
sig <- 0.25
l[1] <- rnorm(n=1, mean = rho * l0, sd = sig)

mu <- c()
mu[1] <- x[1] * (b1 + (b2 * (x[1] >= a1) + b3 * (x[1] <= a2)))

for(j in 2:n){
   l[j] <- rnorm(n = 1, mean = rho * l[j - 1], sd = sig)
   mu[j] <- x[j] * (b1 + (b2 * (x[j] >= a1) + b3 * (x[j] <= a2)))
}
```

```
y <- rpois(n = n, lambda = exp(mu + l))
```

Now we set up and run a nimble model.

```
library(nimble)
```

```
## nimble version 0.11.1 is loaded.
## For more information on NIMBLE and a User Manual,
## please visit http://R-nimble.org.
```

```
##
## Attaching package: 'nimble'
```

```
## The following object is masked from 'package:stats':
##
##     simulate
```

```
poisTSCode <- nimbleCode({
    B1 ~ dnorm(mean=0,sd=100)
    B2 ~ dnorm(mean=0,sd=100)
    B3 ~ dnorm(mean=0,sd=100)
    a1 ~ dunif(0,5)
    d ~ dunif(1,15)
    a2 <- a1 - d
    mu[1] <- X[1] * (B1 + (B2 * (X[1] >= a1) + B3 * (X[1] <= a2)))
    sigma ~ dexp(0.75)
    lambda0 ~ dnorm(mean = 0, sd = 2)
    rho ~ dnorm(0, sd = 2)
    lambda[1] ~ dnorm(rho * lambda0, sd = sigma)
    for (j in 2:Time){
      mu[j] <- X[j] * (B1 + (B2 * (X[j] >= a1) + B3 * (X[j] <= a2)))
      lambda[j] ~ dnorm(rho * lambda[j - 1], sd = sigma)
    }
    for (j in 1:Time){
      Y[j] ~ dpois(exp(mu[j] + lambda[j]))
    }
})

poisTSData <- list(Y = y,
                   X = x)

poisTSConsts <- list(Time = n)

poisTSInits <- list(lambda0 = 0,
                    sigma = 1,
                    rho = 0,
                    a1 = 1,
                    d = 1,
                    B1 = 0,
                    B2 = 0,
                    B3 = 0)

poisTSModel <- nimbleModel(code = poisTSCode,
                           data = poisTSData,
                           inits = poisTSInits,
```

```
                        constants = poisTSConsts)
```

```
## defining model...
```

```
## building model...
```

```
## setting data and initial values...
```

```
## running calculate on model (any error reports that follow may simply reflect missing values in model
## checking model sizes and dimensions... This model is not fully initialized. This is not an error. To
## model building finished.
```

```
#compile nimble model to C++ code-much faster runtime
C_poisTSModel <- compileNimble(poisTSModel, showCompilerOutput = FALSE)
```

```
## compiling... this may take a minute. Use 'showCompilerOutput = TRUE' to see C++ compilation details.
## compilation finished.
```

```
#configure the MCMC
poisTSModel_conf <- configureMCMC(poisTSModel, thin = 1)
```

```
## ===== Monitors =====
## thin = 1: B1, B2, B3, a1, d, sigma, lambda0, rho
## ===== Samplers =====
## RW sampler (982)
##   - B1
##   - B2
##   - B3
##   - a1
##   - d
##   - sigma
##   - lambda[]  (976 elements)
## conjugate sampler (2)
##   - lambda0
##   - rho
```

```
poisTSModel_conf$removeSampler(c("B1", "B2", "B3", "a1", "d"))
poisTSModel_conf$addSampler(target = c("B1", "B2", "B3"),
                            type = "AF_slice")
poisTSModel_conf$addSampler(target = c("a1", "d"),
                            type = "AF_slice")

#build MCMC
poisTSModelMCMC <- buildMCMC(poisTSModel_conf,enableWAIC=F)

#compile MCMC to C++-much faster
C_poisTSModelMCMC <- compileNimble(poisTSModelMCMC, project=poisTSModel)
```

```
## compiling... this may take a minute. Use 'showCompilerOutput = TRUE' to see C++ compilation details.
## compilation finished.
```

```
#number of MCMC iterations
niter <- 2000000

#set seed for replicability
#set.seed(1)

#call the C++ compiled MCMC model
```

```
samples <- runMCMC(C_poisTSModelMCMC, niter=niter)
```

```
## running chain 1...
```

```
## |-------------|-------------|-------------|-------------|
## |-------------------------------------------------------|
```

Now we can check the mcmc chains for convergence after removing the first 900,000 iterations as burn-in,

```
library(coda)
geweke.diag(samples[-c(1:900000),])
```

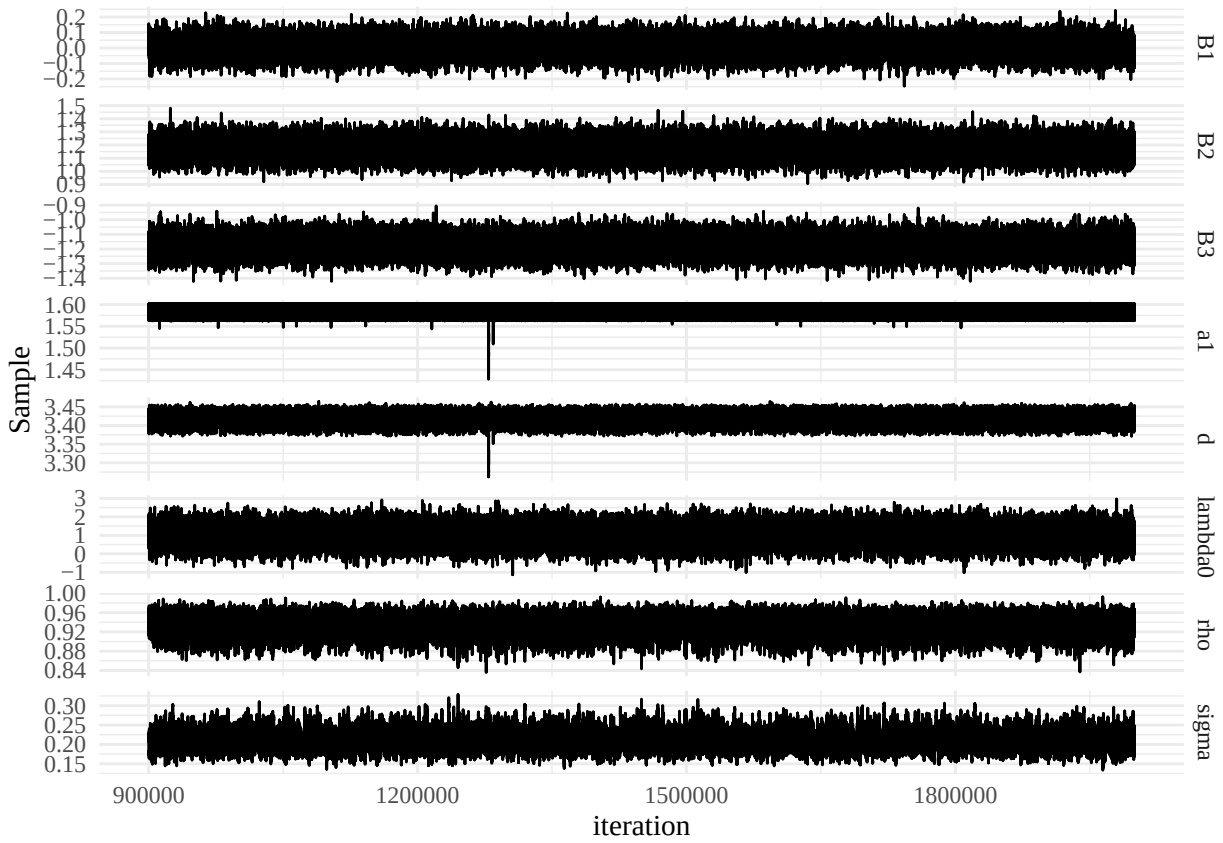
```
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##       B1       B2       B3       a1        d  lambda0      rho    sigma
## -0.85421  0.60073 -0.59638  0.05306 -0.04377  0.16415  0.70743 -1.06906
```

And, finally, plot the results:

```
library(ggplot2)
library(ggpubr)
library(tidyr)

burnin <- 900000
thin <- 10

dim_samples <- dim(samples)
dframe <- as.data.frame(samples[seq(burnin,dim_samples[1],thin),])
iter <- seq(burnin,dim_samples[1],thin)
dframe$iteration <- iter
samples_long <- gather(dframe,
                       key = "Parameter",
                       value = "Sample",
                       -"iteration")
parameters <- c("B1",
                "B2",
                "B3",
                "a1",
                "d",
                "lambda0",
                "rho",
                "sigma")
samples_long$Parameter <- factor(samples_long$Parameter, levels = parameters)
p <- ggplot(data = samples_long) +
        geom_path(mapping = aes(y = Sample, x = iteration)) +
        facet_grid(rows = vars(Parameter), scales = "free") +
        theme_minimal(base_family="serif")
p
```


Sample
B1
B2
B3
a1
d
lambda0
rho
sigma
0.2
0.1
0.0
−0.1
−0.2
1.5
1.4
1.3
1.2
1.1
1.0
0.9
−0.9
−1.0
−1.1
−1.2
−1.3
−1.4
1.60
1.55
1.50
1.45
3.45
3.40
3.35
3.30
3
2
1
0
−1
1.00
0.96
0.92
0.88
0.84
0.30
0.25
0.20
0.15
900000
1200000
1500000
1800000
iteration