BMTRY 719 Lab Course:

Introduction to NIMBLE

Chun-Che Wen

14 March 2024

Table of content

Installation of Nimble package

Installation of Nimble package

- install **nimble** package from CRAN.
- install **Rtools** package (a bit tricky).
  - R version 3.6.3 or lesser (link)
    - using Rtools35.exe
    - make sure to check(✓) the box labelled "Add rtools to system PATH".
  - R version 4.0 or greater (link)
    - using rtools40v2-x86_64.exe (64-bit) or rtools40-i686.exe (32-bit)
    - download **.Renviron** file and save to Documents folder.
    - run code below in R.

```r
writeLines('PATH="${RTOOLS40_HOME}\\usr\\bin;${PATH}"',
           con = "~/.Renviron")
# Restart R again!!!
Sys.which("make") ## "C:\\rtools40\\usr\\bin\\make.exe"
install.packages("jsonlite", type = "source")
```

Introduction to Nimble

Introduction to Nimble

- Combine statistical models in the **BUGS** language from R.
- Compile numerical work in R via C++ without coding any C++.
- Use and customize statistical algorithms (e.g. MCMC)

Winbugs Interface - 1

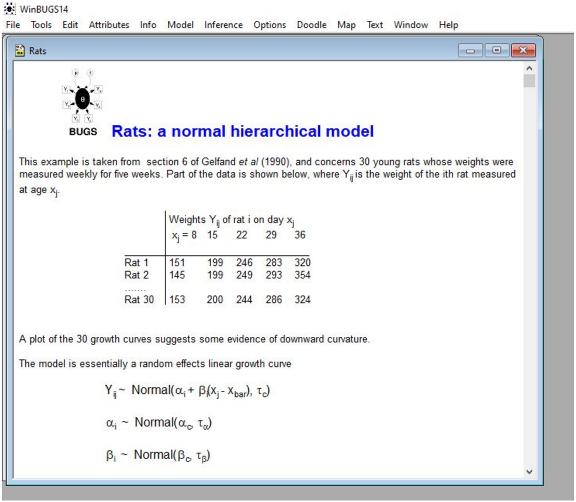

Figure 1: Winbugs Interface

Winbugs Interface - 2

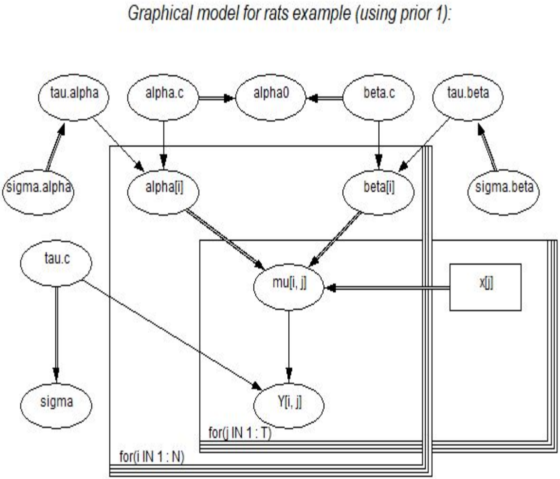

Figure 2: DAGS (Directed Acylic Graphs)

Winbugs Interface - 3

BUGS *language for rats example:*

```
model
{
    for( i in 1 : N ) {
        for( j in 1 : T ) {
            Y[i , j] ~ dnorm(mu[i , j],tau.c)
            mu[i , j] <- alpha[i] + beta[i] * (x[j] - xbar)
        }
        alpha[i] ~ dnorm(alpha.c,tau.alpha)
        beta[i] ~ dnorm(beta.c,tau.beta)
    }
    tau.c ~ dgamma(0.001,0.001)
    sigma <- 1 / sqrt(tau.c)
    alpha.c ~ dnorm(0.0,1.0E-6)
    # Choice of prior of random effects variances
    # Prior 1: uniform on SD
    sigma.alpha~ dunif(0,100)
    sigma.beta~ dunif(0,100)
    tau.alpha<-1/(sigma.alpha*sigma.alpha)
    tau.beta<-1/(sigma.beta*sigma.beta)

    #Prior 2: (not recommended)
    #tau.alpha ~ dgamma(0.001,0.001)
    #tau.beta ~ dgamma(0.001,0.001)

    beta.c ~ dnorm(0.0,1.0E-6)

    alpha0 <- alpha.c - xbar * beta.c
}
```

Figure 3: Bugs Code

Winbugs Interface - 4

```
Data ⇒list(x = c(8.0, 15.0, 22.0, 29.0, 36.0), xbar = 22, N = 30, T = 5,
      Y = structure(
         .Data =   c(151, 199, 246, 283, 320,
                     145, 199, 249, 293, 354,
                     147, 214, 263, 312, 328,
                     155, 200, 237, 272, 297,
                     135, 188, 230, 280, 323,
                     159, 210, 252, 298, 331,
                     141, 189, 231, 275, 305,
                     159, 201, 248, 297, 338,
                     177, 236, 285, 350, 376,
                     134, 182, 220, 260, 296,
                     160, 208, 261, 313, 352,
                     143, 188, 220, 273, 314,
                     154, 200, 244, 289, 325,
                     171, 221, 270, 326, 358,
                     163, 216, 242, 281, 312,
                     160, 207, 248, 288, 324,
                     142, 187, 234, 280, 316,
                     156, 203, 243, 283, 317,
                     157, 212, 259, 307, 336,
                     152, 203, 246, 286, 321,
                     154, 205, 253, 298, 334,
                     139, 190, 225, 267, 302,
                     146, 191, 229, 272, 302,
                     157, 211, 250, 285, 323,
                     132, 185, 237, 286, 331,
                     160, 207, 257, 303, 345,
                     169, 216, 261, 295, 333,
                     157, 205, 248, 289, 316,
                     137, 180, 219, 258, 291,
                     153, 200, 244, 286, 324),
                 .Dim = c(30,5)))⇐
```

Figure 4: Load Data

NIMBLE workflow

- Build the model (**BUGS code**)
- Build the MCMC
  - 2a. **Configure** the MCMC
  - 2b. **Customize** the MCMC
  - 2c. Build the MCMC
- **Compile** the model and MCMC
- Run the MCMC
- Extract the samples

Build the model

Build the model

- Write a BUGS code inside nimbleCode() function.
- Inside the function, only can use BUGS code.
- Note: With a braces{} to include all BUGS code.

```
code <- nimbleCode({
  for(i in 1:n) {y[i] ~ dnorm(beta0+beta1*x[i],
                              sd = sigma)}
  beta0 ~ dnorm(0, sd = 100)
  beta1 ~ dnorm(0, sd = 100)
  sigma ~ dunif(0, 100)
})
```

Build the model

- If we want to specify a distribution, use "$\sim$".
  - $y \sim dnorm(mean, tau)$ (Default is precision)
  - $y \sim dgamma(shape, rate)$
  - $y \sim dbeta(shape1, shpae2)$
  - $y \sim dflat$ (Improper uniform distribution)
- If we want to store values, use "$< -$".

```
code <- nimbleCode({
  tau   ~ dgamma(shape = 0.001, rate = 0.001)
  var   <- 1/tau
  sigma <- sqrt(var)
})
```

Build the model

- Wrong specification - specify twice

```
code <- nimbleCode({
  tau   ~ dgamma(shape = 0.001, rate = 0.001)
  var   ~ dinvgamma(shape = 0.001, rate = 0.001)
})
```

Build the model

- Specify **constants**, **data**, **inits**.
- **NimbleModel()**: check the model specification.

```r
code <- nimbleCode({
  for(i in 1:n) {y[i] ~ dnorm(beta0+beta1*x[i],
                              sd = sigma)}
  beta0 ~ dnorm(0, sd = 100)
  beta1 ~ dnorm(0, sd = 100)
  sigma ~ dunif(0, 100)})
constants <- list(n = n)
data <- list(y = y, x = x)
inits <- list(beta0 = 0, beta1 = 0, sigma = 0.5)
# nimbleModel: check code specification is correct
model <- nimbleModel(code, constants = constants,
                     data = data, inits = inits)
```

- note: constants can't be change after creating a model & data and inits can be changed

Build the model

- If model is correct, see model building finished in the end.

```
> model <- nimbleModel(code, constants = constants, data = data, inits = inits)
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 variables) ...
checking model sizes and dimensions...
model building finished.
```

- Common errors
  - Likelihood function does not make sense (e.g. complicated model).
  - Forget to specify constants or inits.
  - Avoid confused variable names.

Build the MCMC

2a. Configure the MCMC

- Remind of the model

```
code <- nimbleCode({
  for(i in 1:n) {y[i] ~ dnorm(beta0+beta1*x[i],
                               sd = sigma)}
  beta0 ~ dnorm(0, sd = 100)
  beta1 ~ dnorm(0, sd = 100)
  sigma ~ dunif(0, 100)
})
x <- x-mean(x)  # center for better MCMC performance
constants <- list(n = n)
data <- list(y = y,x = x)
inits <- list(beta0=0, beta1=0,sigma=0.5)
```

2a. Configure the MCMC

```r
model <- nimbleModel(code, constants = constants,
                     data = data, inits = inits)
mcmcConf <- configureMCMC(model)
```

```
## ===== Monitors =====
## thin = 1: beta0, beta1, sigma
## ===== Samplers =====
## RW sampler (1)
##   - sigma
## conjugate sampler (2)
##   - beta0
##   - beta1
```

```r
mcmcConf$printSamplers()#Look up sampler assignments.
```

```
## [1] conjugate_dnorm_dnorm_additive sampler: beta0
## [2] conjugate_dnorm_dnorm_linear sampler: beta1
## [3] RW sampler: sigma
```

2a. Configure the MCMC

- Add parameter monitors

```
code <- nimbleCode({
  for(i in 1:n) {y[i] ~ dnorm(beta0+beta1*x[i],
                               sd = sigma)}
  beta0 ~ dnorm(0, sd = 100)
  beta1 ~ dnorm(0, sd = 100)
  sigma ~ dunif(0, 100)
  var   <- pow(sigma,2) # power
  tau   <- 1/sigma})
model <- nimbleModel(code, constants = constants,
                     data = data, inits = inits)
cmodel   <- compileNimble(model)# First compilation
mcmcConf <- configureMCMC(model,print = FALSE)
# add monitor of tau & var
mcmcConf$addMonitors(c("var","tau"))
```

```
## thin = 1: beta0, beta1, sigma, tau, var
```

2b. Customize the MCMC

- Change the samplers for each parameter
- The default is 'RW' which specifies adaptive Metropolis-Hastings sampling with a normal proposal distribution.
- Remove **old** sampler, and then add **new** sampler.
- addsampler(target=c(),type=" ", control=list())

```
# customizing the MCMC
modelConf$removeSampler("sigma")
modelConf$addSampler(target=c("sigma"),type="RW_block",
                     control=list(adaptInterval=100))
```

2c. Build the MCMC

- buildMCMC(): Build a MCMC project
  - optional argument: specify monitors, thin, ...
- compileMCMC(): Compile in C++ for faster execution

```
modelMCMC <- buildMCMC(modelConf)
modelMCMC <- compileNimble(modelMCMC, project = model)
```

Note

- Two compilations when we run the Nimble.
- First, it is used after you specify the model.
  - cmodel <- compileNimble(**model**)
  - Object inside the function compileNimble is “nimbleModel”.
- Second, it is used after you build MCMC.
  - modelMCMC <- buildMCMC(modelConf)
  - modelMCMC <- compileNimble(**modelMCMC**, project = model)
  - Object inside the function compileNimble is “MCMC”

Run the MCMC

- Two functions to run the MCMC.
  - runMCMC() and nimbleMCMC()

```
niter <- 1500
burn <- 0
set.seed(1)
samples <- runMCMC(modelMCMC, niter = niter,
                   nburnin = burn, nchains = 1,
                   WAIC = TRUE, summary=TRUE)
samples <- nimbleMCMCcode(code, constants = constants,
                          data = data, inits = inits,
                          niter = niter,nburnin = burn,
                          nchains = 1,
                          WAIC = TRUE, summary=TRUE)
```

Run the MCMC

```r
niter <- 1500
burn  <- 500
set.seed(1)
samples <- runMCMC(modelMCMC, niter = niter,
                   nburnin = burn, nchains = 1,
                  summary=TRUE)
```

```
running chain 1...
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
>
> samples$summary
           Mean    Median    St.Dev. 95%CI_low 95%CI_upp
beta0 0.4876180 0.4878118 0.04864795 0.3892818 0.5851155
beta1 1.4575167 1.4573899 0.04493596 1.3707490 1.5449565
sigma 1.0605752 1.0600440 0.03351865 0.9974975 1.1299773
tau   0.8916899 0.8899226 0.05627161 0.7831781 1.0050239
var   1.1259422 1.1236933 0.07127340 0.9950012 1.2768488
```

Figure 5: MCMC Result

Extract the samples

```
> head(samples,20)
          beta0    beta1    sigma       tau      var
 [1,] 0.4301937 1.518479 2.119647 0.2225732 4.492904
 [2,] 0.4777968 1.413076 2.126849 0.2210683 4.523488
 [3,] 0.4996612 1.550302 2.095187 0.2278003 4.389810
 [4,] 0.3970649 1.439354 2.095554 0.2277206 4.391346
 [5,] 0.5245498 1.509986 2.082012 0.2306926 4.334773
 [6,] 0.4786887 1.434096 2.092211 0.2284489 4.377346
 [7,] 0.5017511 1.517634 2.177479 0.2109076 4.741413
 [8,] 0.4846811 1.585142 2.114234 0.2237143 4.469986
 [9,] 0.4180356 1.539878 2.018983 0.2453209 4.076293
[10,] 0.4434412 1.536793 2.082583 0.2305660 4.337154
[11,] 0.5258653 1.576032 2.102533 0.2262114 4.420644
[12,] 0.5231422 1.561919 2.089516 0.2290387 4.366075
[13,] 0.4200855 1.459714 2.136624 0.2190503 4.565161
[14,] 0.4022012 1.536679 2.045446 0.2390143 4.183850
[15,] 0.4427369 1.567962 2.062345 0.2351134 4.253267
[16,] 0.4100892 1.534372 2.134347 0.2195178 4.555439
[17,] 0.4668868 1.553205 2.088472 0.2292677 4.361714
[18,] 0.4876299 1.504242 1.974207 0.2565751 3.897494
[19,] 0.4475540 1.554493 2.079064 0.2313472 4.322507
[20,] 0.4011367 1.524864 2.168078 0.2127405 4.700562
```

Figure 6: MCMC Result

Extract the samples

- Demonstrate trace plot, summary table, and WAIC.
- Use **coda** package for MCMC diagnostics in simulation example.
- Use **mcmcplot** package for trace plot/ density plots,. . . etc.

Nimble Demostration

- 1. Simple linear regression
- 2. Logistic regression
- 3. Poisson regression
- 4. Negative-Binomial regression

Next Time

We will focus on advanced modeling:

- 1. Random intercept model
- 2. Random slope model
- 3. Logistic random intercept model
- 4. Logistic random slope model
- 5. Spatial modeling