Chapter 9

Spatial Epidemiology in WinBUGS

Contents

Figures

WinBUGS is specialized program that can incorporate spatial variability in a variety of modeling procedures so the general framework of running a model will be described. While there are numerous concepts for spatial models and alternate ways to get models into WinBUGS (e.g., R2WinBUGS), we will go over the basics of running heirarchical Bayesian models in WinBUGS. Although we will not go over the concepts in detail, this short tutorial should enable a novice to load models and data into the WinBUGS environment. All pertinent data can be prepared in any platform but needs to be presented to WinBUGS in the proper format. If not R then Notepad works well for this as the data needs to be presented as comma-separated values to load the data. WinBUGS requires that each section be highlighted or called in order to perform the components. The code to follow will assist in setting up data for use in WinBUGS but an entire book would be needed to explain Bayesian Hierachical Models so we will not cover the theory here. For those interested, we would recommend attending a workshop and reading several books of varying levels of complexity such as Hierarchical Modeling and Analysis for Spatial Data (Bannerjee et al. 2004), Bayesian Disease Mapping (Lawson 2009), and Applied Spatial Data Analysis with R (Bivand et al. 2008).

Sections 9.1 to 9.3 will detail the code necessary to manipulate all necessary location

and GIS data within R. These sections will enable the user to load in covariate data, extract data from within a sampling gird, and prepare data to be used in WinBUGS or using R2WinBUGS. Sections 9.4 to 9.11 will detail the process of entering the appropriate data directly into WinBUGS provided the adjacency matrix and data is formatted properly.

9.1 Data preparation in R

1. Exercise 9.1 - Download and extract zip folder into your preferred location
2. Set working directory to the extracted folder in R under File - Change dir...
3. First we need to load the packages needed for the exercise

```
library(sp)
library(lattice)
library(rgdal)#readOGR
library(rgeos)#gIntersection
library(raster)#to use "raster" function
library(adehabitatHR)
library(maptools)#readAsciiGrid
library(zoo)
```

4. Now open the script "Elaeo_dataprep.R" and run code directly from the script

```
#Load and clean up the location of samples collected during disease surveillance
#for moose
snowy <-read.csv("SnowySamples.csv", header=T)
str(snowy)

#Clean up by deleting extraneous columns if needed
snowy <- snowy[c(-20:-38, -40:-64)]
snowy$Status <- snowy$E_schneide

#Make a spatial data frame of locations after removing outliers
coords<-data.frame(x = snowy$X_Coordina, y = snowy$Y_Coordina)
utm.crs<-"+proj=utm +zone=13 +datum=NAD83 +units=m +no_defs +ellps=GRS80
    +towgs84=0,0,0"
utm.spdf <- SpatialPointsDataFrame(coords= coords, data = snowy,
    proj4string = CRS(utm.crs))
```

5. We now need to load some raster layers of covariates that may be related to disease occurrence

```
#Load DEM raster layer
dem <-raster("snowydem")
image(dem)
class(dem)
proj4string(dem)

#Now transform projections all to match DEM (i.e., Albers)
Albers.crs <-CRS("+proj=aea +lat_1=20 +lat_2=60 +lat_0=40 +lon_0=-96
     +x_0=0 +y_0=0 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs")
snowy.spdf <-spTransform(utm.spdf, CRS=Albers.crs)
```

6. Now we can create a sampling grid that overlaps our disease locations by getting boundary box information from our locations. We added 3 rows of cells (3610 x 3 = 10830) around our outer most samples to encompass all disease samples and neighboring cells until we can figure out how to expand grid polygons in a simpler way. Alternatively, simply use the coordinates from the boundary box (bbox code) of your locations to create your sampling grid.

```
sublette.df <- as.data.frame(sublette.spdf)
str(sublette.df)
minx <- (min(sublette.df$x)-10830)
maxx <- (max(sublette.df$x)+10830)
miny <- (min(sublette.df$y)-10830)
maxy <- (max(sublette.df$y)+10830)

## create vectors of the x and y points
x <- seq(from = minx, to = maxx, by = 3610)
y <- seq(from = miny, to = maxy, by = 3610)

#Alternate bbox code for spatial points
#       min       max
#x -854784.4 -724665.0
#y  156859.0  247343.2

## create vectors of the x and y points
#x <- seq(from = -854784.4, to = -724665.0, by = 3610)
#y <- seq(from = 156859.0, to = 247343.2, by = 3610)

## create a grid of all pairs of coordinates (as a data.frame)
xy <- expand.grid(x = x, y = y)
class(xy)
str(xy)

#Identifiy projection before creating Spatial Points Data Frame
Albers.crs2 <-"+proj=aea +lat_1=20 +lat_2=60 +lat_0=40 +lon_0=-96
    +x_0=0 +y_0=0 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"

#NOTE: Albers.crs2 is needed because SPDF needs different projection command
#than spTransform above

grid.pts<-SpatialPointsDataFrame(coords= xy, data=xy,
     proj4string = CRS(Albers.crs2))
proj4string(grid.pts)
plot(grid.pts)
gridded(grid.pts)
class(grid.pts)

#Need to define points as a grid to convert to Spatial Polygons below
gridded(grid.pts) <- TRUE
gridded(grid.pts)
str(grid.pts)
plot(grid.pts)
```

```
#Convert grid points to Spatial Polygons in essence converting to a shapefile
gridsp <- as(grid.pts, "SpatialPolygons")
str(gridsp)
plot(gridsp)
class(gridsp)
summary(gridsp)
```

7. Now convert gridpts to Spatial Polygons Data Frame for added flexibility in manipulating layer

```
grid <- SpatialPolygonsDataFrame(gridsp, data=data.frame(id=row.names(gridsp),
     row.names=row.names(gridsp)))
class(grid)
plot(grid)
names.grd<-sapply(grid@polygons, function(x) slot(x,"ID"))
text(coordinates(grid), labels=sapply(slot(grid, "polygons"),
     function(i) slot(i, "ID")), cex=0.3)

#Let's check to see if all grid cells are the same size?
summary(grid)
getSlots(class(grid))
class(slot(grid, "polygons")[[1]])
getSlots(class(slot(grid, "polygons")[[1]]))

#Check area of each cell in the sampling grid in square meters
sapply(slot(grid, "polygons"), function(x) slot(x,"area"))
#[1] 13032100

#Grid cell size converted strto square kilometers
13032100/1000000
#[1] 13.0321 is grid cell size in square kilometers
```

9.2 Raster manipulation in R

8. Continue with same script loaded after finishing Exercise 9.1

9. We loaded moose sample locations and created our sampling grid above so now it is time to work with covariate data that was provided in various forms. We imported DEM in the last exercise because we needed to determine the projection information early on to prepare our grid. It is easier to project moose data to fit a raster projection than vice versa so now let's continue adding additional covariates from raster data.

```
#The layer below is a mule deer HSI raster layer without disturbance from based
#on data from Sawyer et al. 2009 incorporated into a layer because mule deer are
#considered host for the parasite we are investigating
nodis <-raster("snowynodis")
nodis
plot(nodis)
summary(nodis)

#Need to remove NoData from mule deer HSI layer
nodis[is.na(nodis[])] <- 0
```

10. Using functions from the *raster* package, we can calculate slope and aspect from DEM layer imported above

```
slope = terrain(dem,opt='slope', unit='degrees')
aspect = terrain(dem,opt='aspect', unit='degrees')
dem #Now let's see metadata for each layer
slope
aspect

plot(dem)
plot(grid, add=T)
```

11. We also want to look at Land Cover data for this region and reclassify it into fewer categories for comparison and manipulation

```
nlcdall <- raster("nlcd_snowy")
nlcdall #Look at raster values for the habitat layer
#Values range from 11 to 95

#Or plot to visualize categories in legend
plot(nlcdall)

#Reclassify the values into 7 groups
#all values between 0 and 20 equal 1, etc.
m <- c(0, 19, 1, 20, 39, 2, 40, 50, 3, 51,68, 4, 69,79, 5, 80, 88, 6, 89, 99, 7)
rclmat <- matrix(m, ncol=3, byrow=TRUE)
rc <- reclassify(nlcdall, rclmat)
plot(rc) #Now only 7 categories
rc #Now only 7 categories
class(rc)

#Check to be sure all raster have same extent for Stack creation
compareRaster(dem,slope,aspect,nodis,rc)
#[1] TRUE
```

12. Minimize the size of the data for demonstartion purposes

```
##########################################################################
##########################################################################
##NOTE: Code in this box was simply for demonstration purposes to reduce overall
## time for processing during class. Skip this section of code if using your
## own data and your computer has the appropriate processing capabilities.

#First we will clip out the raster layers by zooming into only a few locations
plot(rc)
plot(grid, add=T)
points(snowy.spdf)
#Code below is used to just zoom in on grid using raster layer
e <- drawExtent()
#click on top left of crop box and bottom right of crop box create zoom
newclip <- crop(rc,e)
plot(newclip)
plot(grid, add=T)
points(snowy.spdf, col="red")
```

```
#Clip locations within extent of raster
samp_clip <- crop(snowy.spdf,newclip)
plot(newclip)
plot(samp_clip, add=T)
grid_clip <- crop(grid, newclip)
plot(grid_clip, add=T)
slope2 <- crop(slope,newclip)
aspect2 <- crop(aspect,newclip)
dem2 <- crop(dem,newclip)
HSI <- crop(nodis, newclip)

#Check to be sure all rasters have same extent for Stack creation
compareRaster(dem2,slope2,aspect2,HSI,newclip)
#[1] TRUE

grid <- grid_clip #rename clipped grid as grid to match code below
rc <- newclip #rename clipped Land Cover as "rc" to match code below
snowy.spdf <- samp_clip

#Create a Stack of all Raster layers
#This will take a long long time if rasters have a large extent
r <- stack(list(dem=dem2, slope=slope2, aspect=aspect2, "mule deer HSI"=HSI,
     nlcd=newclip))

#END Demonstration code
##########################################################################
##########################################################################
```

13. Now we want to combine all raster layers into a multi-layered raster called a "stack" below before proceeding if using your own data. Otherwise skip Line 13 if using demonstration code above and continue on with Line 14

```
#Create a Stack of all Rasters
#This will take a long long time if rasters have a large extent
r <- stack(list(dem=dem, slope=slope, aspect=aspect, "HSI"=nodis, nlcd=rc))

nlayers(r) #Show how many layers are in the "r" stack
plot(r) #Visualize "r"
names(r) #Names of each raster in the "r" stack
```

9.3 Combine data formats in R

14. Continue with same script loaded after finishing Exercise 9.1 and 9.2

15. Here we need to get a mean for each covariate for each 13 km^2 cell of our sampling grid

```
#Extracts all rasters within each sampling grid cell
ext <- extract(r, grid, weights=TRUE, fun = mean)

head(ext) #view the results
```

```
#      dem       slope     aspect   mule.deer.HSI   nlcd
#[1,] 2730.461  9.801106 227.6434    0.28312888  3.202686
#[2,] 2661.003 11.862190 120.0121    0.05178489  3.195367
#[3,] 2443.063  1.644629 136.0088    0.01859540  3.989930
#[4,] 2450.666  8.212403 199.1163    0.23681863  3.837849
#[5,] 2570.362  8.625199 212.5139    0.26878506  3.333714
#[6,] 2685.348  5.395405 205.6125    0.22692773  3.086528

#############################################################################
#NOTE above that for each grid cell in the sampling grid layer (i.e., grid),
# the "extract" function resulted in means for dem, slope, aspect, and HSI
#for all sampling grid cells but "nlcd" (last column of data)  resulted in mean
# cover categories so need to run nlcd separate with more appropriate code below.
#############################################################################
```

16. Mean land cover category is not appropriate here so we need to handle Land Cover layer (i.e., rc) separately

```
#Code below extracts nlcd by land cover category and determines how many
#cells of each type were in each sampling grid.
ovR = extract(rc,grid, byid=TRUE)
head(ovR)

#Summarize results by sampling grid ID
#Land Cover category and number of cells (30x30m raster cells)
tab <- lapply(ovR,table)
tab[[1]]
# 3
#18

tab[[48]]
#   3    4    5    7
#6618   80   95  167

##########################################
#Code here thanks to Tyler Wagner, PA Coop Unit, for creating this loop
#to summarize proportions of habitat within each grid cell
##########################################
# Created land use categories
lus <- 1:7

##### Loop through and append missing land use categories to each grid cell
ovRnew <- list()
for(i in 1:length(grid)[1] ){
# Land use cats in a given cell
temp1 <- unique(ovR[[i]])
# Give missing category 999 value
ma1 <- match(lus, temp1, nomatch = 999, incomparables = NULL)
# Get location (category of missing land use type)
miss <- which(ma1%in%999)
ovRnew[[i]] <- c(ovR[[i]], miss)
```

```
}

# New summary of land use in a grid cell
tab2 <- lapply(ovRnew,table)
tab2[[1]]
tab[[1]]

# Proportions of all land cover types per grid cell
prop <- list()
for(i in 1:length(grid)[1] ){
prop[[i]] <- round((margin.table(tab2[[i]],1)/margin.table(tab2[[i]])),digits = 6)
}

#Function coredata is from the zoo package to convert the
#the proportions from a list to a matrix
M <- coredata(do.call(cbind, lapply(prop, zoo)))
colnames(M) <- NULL

#Transpose matrix so land cover become separate columns of data
matrix <- t(M)
matrix

#Now convert the matrix to a data frame so it is easier to manipulate
dfland <- as.data.frame(matrix)

#Assigning column names to land cover
colnames(dfland) <- c("water","developed","forest","shrub","grass","crop","wetland")
dfland[1:5,]
```

17. Now that we have Land Cover in a similar format as the DEM-derived data, we want to convert ext (the combined extracted rasters) into a data frame so it is easier to manipulate as well. The "extract" function in the *raster* package is supposed to be able to do this but does not work for some reason.

```
elecov <- as.data.frame(ext)
head(elecov)
```

18. To bring all of these datasets together for Bayesian hierarchical modeling, we need to assign sampling grid identification numbers to each moose based on where it was harvested. We also need to assign sampling grid cell identification for the covariate summaries so we can join all into one master data set

```
#Plot and look at sampling grid ID
plot(grid)
text(coordinates(grid), labels=sapply(slot(grid, "polygons"), function(i)
     slot(i, "ID")), cex=0.8)

#Now assign actual sampling grid IDs to the covariate summaries in "elecov"
elecov$id <- paste(grid@data$id)#, function(x) slot(x,"ID"))
demdata <- elecov[-c(5)]#rename to cleanup by removing incorrect nlcd column
head(demdata)

#We can now combine dem data with Land Cover data to get our master dataset for
```

```
#each sampling grid cell
data <- cbind(demdata, dfland) # rbind list elements
head(data)
data$GRID <- data$id #This is mainly just to a check on matching grid IDs for later
data[1:10,]
```

19. We also need to identify the sampling grid cell that each moose was located and match the moose up with its respective covariate data based on grid cell ID

```
#First let's assign the grid cell ID to each moose sampled
snowy2 <- over(snowy.spdf,grid)
snowy2[1:5,]
#Now add column to moose demographic data identifying sampling grid cell ID
new <- cbind(snowy.spdf@data,snowy2)
new[1:5,]

#Now we need to "join" the appropriate covariate data to each moose sampled based on
#the sampling grid cell it occurred in (e.g., g953)
data[1:5,]
data <- data[-c(13)]#remove duplicate id column or program throws an error
mydata <- merge(new, data, by=c("id"))
mydata[1:10,]

#Save data to use in exercise later
write.table(mydata,"SnowyData.txt", sep="\t")

#Save workspace so all data is available
save.image("Epi_dataprep.RData")
```

9.4 Data preparation of NDVI covariate

Normalized Difference Vegetation Index (NDVI) is a satellite-derived global vegetation indicator based on vegetation reflectance that proivdes information on vegetation productivity and phenology (Hamel et al. 2009). NDVI data comes in 15-day composite images, values range from 0.0 to 1.0, and data manipulation is required to identify the parameter of interest (e.g., peak timing of high quality vegetation). For our purpose, we will determine maximum increase between successive NDVI sampling periods and the sum of bimonthly values for each month they area available (i.e., May, July, September) similar to previous research (Hamel et al. 2009)

1. Exercise 9.4 - Download and extract zip folder into your preferred location
2. Set working directory to the extracted folder in R under File - Change dir...
3. First we need to load the packages needed for the exercise

```
library(sp)
library(lattice)
library(rgdal)#readOGR
library(rgeos)#gIntersection
library(raster)#to use "raster" function
library(adehabitatHR)
library(maptools)#readAsciiGrid
```

```
library(zoo)
```

4. Now open the script "SnowyNDVIprep.R" and run code directly from the script

5. We will start by importing individual rasters of our study site for each time period NDVI was available. We will then combine all layers into a raster "stack" for ease of manipulation.

```
ndviMay09 <- raster("snowyMay09.tif")
ndviMay25 <- raster("snowyMay25.tif")
ndviJune10 <- raster("snowyJun10.tif")
ndviJuly12 <- raster("snowyJuly12.tif")
ndviJuly28 <- raster("snowyJuly28.tif")
ndviAug29 <- raster("snowyAug29.tif")
ndviSep14 <- raster("snowySep14.tif")
ndviSep30 <- raster("snowySep30.tif")
proj4string(ndviMay09)

#Create a Stack of all Rasters
r <- stack(list(ndviMay09=ndviMay09, ndviMay25=ndviMay25, ndviJune10=ndviJune10,
    ndviJuly12=ndviJuly12,ndviJuly28=ndviJuly28,ndviAug29=ndviAug29,
    ndviSep14=ndviSep14, ndviSep30=ndviSep30))
nlayers(r)
plot(r)
names(r)
```

6. Then we can get a mean for each NDVI layer for each 13 km2 cell of our sampling grid by extracting all rasters within each sampling grid cell. We will start first by creating our own functions using the raster package.

```
#We will start by creating a function to determine
#maximum increase between successive NDVI periods
#NOTE: x[1] refers to the first layer in your raster stack

maxmay <- function(x){x[1]-x[2]}
maxjune <- function(x){x[2]-x[3]}
maxjuly1 <- function(x){x[3]-x[4]}
maxjuly2 <- function(x){x[4]-x[5]}
maxaug <- function(x){x[5]-x[6]}
maxsept1 <- function(x){x[6]-x[7]}
maxsept2 <- function(x){x[7]-x[8]}


#Now perform the function on each raster in the stack
may <- calc(r,maxmay)
june <- calc(r,maxjune)
july1 <- calc(r,maxjuly1)
july2 <- calc(r,maxjuly2)
aug <- calc(r,maxaug)
sept1 <- calc(r,maxsept1)
sept2 <- calc(r,maxsept2)
```

7. We can plot out each layer in 4 x 3 dimensions if we want to look over what the function created and determine values that resulted from performing the functions on each raster

group.

```
#Set up the figure layout
par(mfcol=c(3,3),mar=c(2,3.5,2,2),oma=c(3,3,3,3)) #Bottom,Left,Top,Right.

plot(may)
# Create a title with a red, bold, italic font
title(main="May", col.main="black", font.main=4)

plot(june)
# Create a title with a red, bold, italic font
title(main="June", col.main="black", font.main=4)

plot(july1)
# Create a title with a red, bold, italic font
title(main="July 1", col.main="black", font.main=4)

plot(july2)
# Create a title with a red, bold, italic font
title(main="July 2", col.main="black", font.main=4)

plot(aug)
# Create a title with a red, bold, italic font
title(main="Aug", col.main="black", font.main=4)

plot(sept1)
# Create a title with a red, bold, italic font
title(main="Sept 1", col.main="black", font.main=4)

plot(sept2)
# Create a title with a red, bold, italic font
title(main="Sept 2", col.main="black", font.main=4)
```

8. Now we can use the raster package to create functions to determine sum of the bimonthly values for May, July, and September

```
#Start by creating a function to sum the bimonthly values for May, July,
#and September
summay <- function(x){x[1]+x[2]}
sumjuly <- function(x){x[4]+x[5]}
sumsept <- function(x){x[7]+x[8]}

#Now perform the function on each month that we have 2 layers of NDVI
maysum <- calc(r,summay)
julysum <- calc(r,sumjuly)
septsum <- calc(r,sumsept)
```

9. Plot out each layer to look over what the function created and determine values that resulted from performing the functions for each month.

```
windows()#opens a new graphic window if needed
par(mfcol=c(2,2))

plot(maysum)
```

```
# Create a title with a red, bold, italic font
title(main="May", col.main="black", font.main=4)
plot(julysum)

# Create a title with a red, bold, italic font
title(main="July", col.main="black", font.main=4)

plot(septsum)
# Create a title with a red, bold, italic font
title(main="Sept", col.main="black", font.main=4)
```

10. Now we need to get a mean for each covariate for each 13 km^2 cell of our sampling grid similar to our DEM layer means

```
#Means for maximum increase
extmay <- extract(may, grid, weights=TRUE, fun = mean)
extjune <- extract(june, grid, weights=TRUE, fun = mean)
extjuly1 <- extract(july1, grid, weights=TRUE, fun = mean)
extjuly2 <- extract(july2, grid, weights=TRUE, fun = mean)
extaug <- extract(aug, grid, weights=TRUE, fun = mean)
extsept1 <- extract(sept1, grid, weights=TRUE, fun = mean)
extsept2 <- extract(sept2, grid, weights=TRUE, fun = mean)

#Means for bimontly sums
extmaysum <- extract(maysum, grid, weights=TRUE, fun = mean)
extjulysum <- extract(julysum, grid, weights=TRUE, fun = mean)
extseptsum <- extract(septsum, grid, weights=TRUE, fun = mean)
```

11. Now convert each matrix to a data frame so it is easier to manipulate then combine into a single dataset

```
mayNDVImax <- as.data.frame(extmay)
juneNDVImax <- as.data.frame(extjune)
jul1NDVImax <- as.data.frame(extjuly1)
jul2NDVImax <- as.data.frame(extjuly2)
augNDVImax <- as.data.frame(extaug)
sep1NDVImax <- as.data.frame(extsept1)
sep2NDVImax <- as.data.frame(extsept2)

#Means for bimontly sums
mayNDVIsum <- as.data.frame(extmaysum)
julyNDVImax <- as.data.frame(extjulysum)
sepNDVImax <- as.data.frame(extseptsum)

#Bind all NDVI layers created above
Snowy_NDVI <- cbind(maySnowymax,juneSnowymax,jul1Snowymax,jul2Snowymax,
   augSnowymax,sep1Snowymax,sep2Snowymax,maySnowysum,julySnowysum,sepSnowysum)
colnames(Snowy_NDVI) <- c("maySnowymax","juneSnowymax","jul1Snowymax",
   "jul2Snowymax","augSnowymax","sep1Snowymax","sep2Snowymax","maySnowysum",
   "julySnowysum","sepSnowysum")
head(Snowy_NDVI)

#Now assign actual sampling grid IDs to the covariate summaries in "Snowyl_NDVI"
```

```
Snowy_NDVI$id <- paste(snowygrid@data$id)
head(Snowy_NDVI)

write.table(Snowy_NDVI,"SnowyNDVI.txt", sep="\t")

#Save workspace so all data is available
save.image("Snowy_NDVI.RData")
```

12. Now let's combine Habitat and DEM data from previous exercise to NDVI data just created.

```
#First let's import the previous NLCD and DEM data we created
snowy2 <- read.csv("SnowyData.txt", sep="\t")
snowy2[1:5,]

#Now we need to "join" the appropriate covariate data to each moose sampled
#based on the sampling grid cell it occurred in (e.g., g953)
Snowy_NDVI[1:5,]
SnowyFinal <- merge(snowy2, Snowy_NDVI, by=c("id"))
SnowyFinal[1:10,]

write.table(SnowyFinal,"SnowyFinal.txt", sep="\t")
#Copy file into R2WinBUGS exercise 9.5
```

9.5 Data preparation for R2WinBUGS

1. Exercise 9.5 - Download and extract zip folder into your preferred location
2. Set working directory to the extracted folder in R under File - Change dir...
3. First we need to load the packages needed for the exercise

```
library(R2WinBUGS)
library(sp)
library(spdep)
library(rgdal)
```

4. Now open the script "SnowyR2winbugs.R" and run code directly from the script
5. Import dataset created previously

```
df <- read.table("SnowyFinal.txt", sep="\t")
head(df)
str(df)
df$IDvar <- substr(df$id, 2,5)
df$IDvar <- as.integer(df$IDvar)

#Clean up and remove missing data for disease status, sex, and age
summary(df$Status)
df <- subset(df, df$Status !="Unknown")
df$Status <- factor(df$Status)
summary(df$Status)

#Recode Sex classes and remove NAs
```

```
summary(df$Sex)
df <- subset(df, df$Sex !="")
df$Sex <- factor(df$Sex)
df$Sex2 <- as.character(df$Sex)
df$Sex2[df$Sex2 == "Male"] <- "1"
df$Sex2[df$Sex2 == "Female"] <- "0"
df$Sex2

#Recode Age classes and remove NAs
summary(df$Age)
df <- subset(df, df$Age !="")
df$Age <- factor(df$Age)
table(df$Age)
#Age classes
# 2-5        3       6+    Adult     Calf Yearling
# 188        1       79        5       36       30

#Combine ages classes
df$NewAge <- df$Age
levels(df$NewAge)<-list(Yearling=c("Calf","Yearling"),Adult=c("2-5","3","6+",
     "Adult"))
summary(df$NewAge)

#Now convert age to numeric with Yearling=0 (baseline) and Adult=1
df$Age2 <- df$NewAge
df$Age2 <- as.character(df$Age2)
df$Age2[df$Age2 == "Adult"] <- "1"
df$Age2[df$Age2 == "Yearling"] <- "0"
df$Age2

df$KillYear <- as.factor(df$KillYear)

summary(df$KillYear)
#2009 2011 2012
  23   15    1
```

6. Now we are going to re-create our spatial grid used in the previous code. Alternatively, we could export the grid as a shapefile and import it here but this is preferable so sampling grid cell IDs match up

```
#Make a spatial data frame of locations after removing outliers
coords<-data.frame(x = df$X_Coordina, y = df$Y_Coordina)
utm.crs<-"+proj=utm +zone=12 +datum=NAD83 +units=m +no_defs +ellps=GRS80
     +towgs84=0,0,0"
utm.spdf <- SpatialPointsDataFrame(coords= coords, data = df, proj4string =
    CRS(utm.crs))

#Now transform projections all to match DEM (i.e., Albers)
Albers.crs <-CRS("+proj=aea +lat_1=20 +lat_2=60 +lat_0=40 +lon_0=-96 +x_0=0
    +y_0=0 +ellps=GRS80
    +towgs84=0,0,0,0,0,0,0 +units=m +no_defs")
df.spdf <-spTransform(utm.spdf, CRS=Albers.crs)
```

```
#NOTE: We added 3 cells around outer most samples to encompass all disease
#samples until can figure out how to expand grid polygons
snowy.df <- as.data.frame(df.spdf)
str(snowy.df)
minx <- (min(snowy.df$x)-10830)
maxx <- (max(snowy.df$x)+10830)
miny <- (min(snowy.df$y)-10830)
maxy <- (max(snowy.df$y)+10830)

## create vectors of the x and y points
x <- seq(from = minx, to = maxx, by = 3610)
y <- seq(from = miny, to = maxy, by = 3610)

## create a grid of all pairs of coordinates (as a data.frame)
xy <- expand.grid(x = x, y = y)

#Identifiy projection before creating Spatial Points Data Frame
Albers.crs2 <-"+proj=aea +lat_1=20 +lat_2=60 +lat_0=40 +lon_0=-96 +x_0=0
    +y_0=0 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"
#NOTE: Albers.crs2 is needed because SPDF needs different projection command
#than spTransform above
grid.pts<-SpatialPointsDataFrame(coords= xy, data=xy, proj4string =
    CRS(Albers.crs2))
#Need to define points as a grid to convert to Spatial Polygons below
gridded(grid.pts) <- TRUE
#Convert grid points to Spatial Polygons in essence converting to a shapefile
gridsp <- as(grid.pts, "SpatialPolygons")

#Now convert gridpts to Spatial Polygons Data Frame for added flexibility in
#manipulating layer
grid <- SpatialPolygonsDataFrame(gridsp, data=data.frame(id=row.names(gridsp),
      row.names=row.names(gridsp)))
class(grid)
plot(grid)
```

7. Compute adjacency matrix and adj, N, sumNumNeigh required by car.normal function

```
shape_nb <- poly2nb(grid)
NumCells= length(shape_nb)
num=sapply(shape_nb,length)
adj=unlist(shape_nb)
sumNumNeigh=length(unlist(shape_nb))
```

8. Run correlation on covariates to prevent modeling similar covariates

```
rcorr.adjust(df[,c("Status","water","developed","wetland","forest","shrub",
    "grass","crop","HSI","dem","slope","aspect","ndviMay09","ndviMay25",
    "ndviJune10","ndviJuly12", "ndviJuly28","ndviAug29","ndviSep14",
    "ndviSep30"")], type="pearson")
```

9. Prepare values for model inputs

```
Result<-df$Status2
```

```
Grid_ID<-df$IDvar
Sex<-df$Sex2
Age<-df$Age2
Dem <- df$dem
Slope<-df$slope
Aspect <- df$aspect
HSI <- df$HSI
Wat<-df$water
Dev<-df$developed
For<-df$forest
Shru<-df$shrub
Gras<-df$grass
Crop<-df$crop
Wet<-df$wetland
Maymax <- df$mayBigmax
Junmax <- df$juneBigmax
Jul1max <- df$jul1Bigmax
Jul2max <- df$jul2Bigmax
Augmax <- df$augBigmax
Sep1max <- df$sep1Bigmax
Sep2max <- df$sep2Bigmax
Maysum <- df$mayBigsum
Julsum <- df$julyBigsum
Sepsum <- df$sepBigsum
```

10. Define the model in BUGS language

```
sink("sublettepriors.bug")
cat("

model
{

#Priors for CAR model spatial random effects:

b.car[1:NumCells] ~ car.normal(adj[], weights[], num[], tau.car)
for (k in 1:sumNumNeigh)

{
      weights[k] <- 1
}

for (j in 1:NumCells)
{
epsi[j] ~ dnorm(0,tau.epsi)
}

#Other priors
alpha ~ dflat()
beta1 ~ dnorm(0,1.0E-5)
beta2 ~ dnorm(0,1.0E-5)
beta3 ~ dnorm(0,1.0E-5)
```

```
beta4 ~ dnorm(0,1.0E-5)
beta5 ~ dnorm(0,1.0E-5)
beta6 ~ dnorm(0,1.0E-5)
beta7 ~ dnorm(0,1.0E-5)
beta8 ~ dnorm(0,1.0E-5)
beta9 ~ dnorm(0,1.0E-5)
tau.car ~ dgamma(1.0,1.0)
tau.epsi ~ dgamma(17.0393,4.1279)

sd.car<-sd(b.car[])
sd.epsi<-sd(epsi[])
lambda <- sd.car/(sd.car+sd.epsi)

for (i in 1 : 339)
{
Result[i] ~ dbern(pi[i])
logit(pi[i]) <- mu[i]
mu[i] <- alpha + beta1*Sex[i] + beta2*Age[i] + beta3*Dem[i] + beta4*Slope[i]
     + beta5*Aspect[i] + beta6*HSI[i] + beta7*Dev[i] + beta8*For[i]
     + beta9*Gras[i] + b.car[Grid_ID[i]] + epsi[Grid_ID[i]]
}
} # end model
", fill=TRUE)
sink()

# Bundle data
bugs.data <- list(Result=Result, Grid_ID=Grid_ID, NumCells=NumCells,
     sumNumNeigh=sumNumNeigh, num=num, adj=adj, Sex=Sex, Age=Age,
     Dem=Dem, Slope=Slope, Aspect=Aspect, HSI=HSI, Dev=Dev, For=For, Gras=Gras)
```

11. Load initial values

```
inits1<- list(alpha = 0, beta1 = 0,  beta2 = 0, beta3 = 0, beta4 = 0, beta5 = 0,
     beta6 = 0, beta7 = 0, beta8 = 0, beta9 = 0)
inits2<- list(alpha = 0, beta1 = 0,  beta2 = 0, beta3 = 0, beta4 = 0, beta5 = 0,
     beta6 = 0, beta7 = 0, beta8 = 0, beta9 = 0)
inits3<- list(alpha = 0, beta1 = 0,  beta2 = 0, beta3 = 0, beta4 = 0, beta5 = 0,
     beta6 = 0, beta7 = 0, beta8 = 0, beta9 = 0)

inits<- list(inits1, inits2, inits3)

# Paramters to estimate and keep track of
parameters <- c("alpha","beta1","beta2", "beta3", "beta4", "beta5", "beta6",
      "beta7", "beta8", "beta9", "lambda")

# MCMC settings
niter <- 150000
nthin <- 20
nburn <- 10000
nchains <- 3
```

12. Locate WinBUGS by setting path below specifically for the computer used.

```
bugs.dir<-"C:\\Program Files\\WinBUGS14"

# Do the MCMC stuff from R
out <- bugs(data = bugs.data, inits = inits, parameters.to.save = parameters,
model.file = "subletteprior s.bug", n.chains = nchains, n.thin=nthin, n.iter=niter,
n.burnin=nburn, debug=TRUE, bugs.directory=bugs.dir)


print(out, 3)
```

9.6 Data preparation within ArcMap

This section does not have an exercise becasuse we are unable to provide this data due to an agreement with collaborators on this project. This is just a small tutorial on how to prepare data within ArcMap and some considerations for spatial epidemiology. A previous study on bovine tuberculosis (TB) in the northern lower peninsula of Michigan used multivariate conditional logistic regression in a case-control design (Kaneene et al. 2002). Kaneene et al. (2002) used 18 covariates and deer TB prevalence summarized in 3 x 3 blocks (~23 km2) in an area surrounding farms that resulted in P-values and Odds Ratios of risk of disease. Note that this design does not borrow from strength or knowledge of data from adjacent areas.

An alternative way is to link the disease status (positive or negative) of each farm sampled to some landscape-level predictors. This is a multi-step process that can be done in WinBUGS with data prepared in R or ArcMap depending on your level of experience or comfort with either program. There are 3 major considerations to approaching spatial epidemiology that was used in a study on bovine tuberculosis on cattle farms prepared in ArcMap that is the basis for this section (Walter et al. 2014):

1. Spatial Resolution

   First we overlayed a 5 x 5 km grid having a resolution of 25 km^2, which is approximately equal to a quarter township in size. We selected quarter townships as the proper resolution given that township would likely be too coarse a scale and section would be too fine a resolution for model convergence based on previous research with Bayesian hierarchical models (Farnsworth et al. 2006, Walter et al. 2011). This would result in a total of 368 cells covering the Modified Accredited Zone (MAZ; 5 counties) in Michigan and we can then assign the value associated with each landscape-level predictor variable to a farm in our study based on the grid cell that an individual farm was sampled from; thus, all farms sampled from within a particular grid cell were assigned the same value for each landscape-level predictor.

2. Covariates

   It is very important that covariates are based on some *a priori* knowledge of factors contributing to an increase in risk for infection of disease. To simply data dredge and hope some covariates are contained within the top model(s) is wrong and a study should not be designed this way. Researchers designing a study on spatial epidemiology should consider the demographic variables of the host and/or reservoir and well as any environmental or landscape variables that may influence host/reservoir distribution in the landscape. To simply include elevation, slope, and aspect because previous researchers included them is simply incorrect and should be avoided.

3. Distribution of data

The spatial extent of the data across the study area of interest is also of importance due to limitations in computer processors. If the spatial resolution is small and the extent is large and results in >2000 cells across your study regoin, it may take weeks to run models or models may not run at all. The spatial extent of the data that would be suitable to achieve objectives of the study should be determined prior to initiating studies using Bayesian hierarchical modeling in WinBUGS.

9.6.1 Adjacency matrices with weights = 1

Spatial resolution can be handled and incorporated into modeling efforts using Intrinsic Gaussian Conditional Autoregressive Models (ICAR) in WinBUGS using:

```
car.normal(adj[], weights[], num[], tau)
```

where:

Adjacency - a vector listing the ID numbers of the adjacent areas for each area (this can be generated using the Adjacency Tool in R or ArcMap)

Number - A vector of length N (the total number of areas) giving the number of neighbors for each area

Weights - A vector the same length as adj[] giving unnormalized weights associated with each pair of areas

Thus, the random effect of the jth grid cell is conditional on the values of its (usually = 8) neighboring cells. Adjacency matrices were created with the Adjacency for WinBUGS Tool that provides a matrix relating one areal unit to a collection of neighboring areal units in text files for use in WinBUGS (Fig. 7.1). In ArcMap, an adjacency matrix can be created by installing a Toolbox created by the USGS that will result in 3 separate textfiles. Results of these textfiles can be used within your program to run models in WinBUGS.

1. Install Adjacency for WinBUGS Tool and follow program page for setup.
2. Create the adjacency matrix in the GUI that will result in 4 text files although we will only need to use first 2 in our models:
   (a) *Adj.txt* identifies each cell by unique ID that is adjacent to cell 1, cell 2, cell 3, etc., in sequential order (NOTE: Cell ID is not in file, only IDs of adjacent cells)

```
2,3,4,40,
1,3,4,5,8,39,40,44,
1,2,4,5,8,
1,2,3,
2,3,6,7,8,39,43,44,
5,7,8,9,12,43,44,48,
5,6,8,9,12,
```

   (b) *Num.txt* identifies the number of neighbors for each cell in Adj.txt

```
4
8
5
3
8
8
```

5

(c) *Raw.txt* is similar to Adj.txt with the exception that the first number refers to the cell ID that the neighboring cells are adjacent to.

(d) *SumNumNeigh.txt* shows the overall numbers of neighbors that will be manually entered into WinBUGS code.

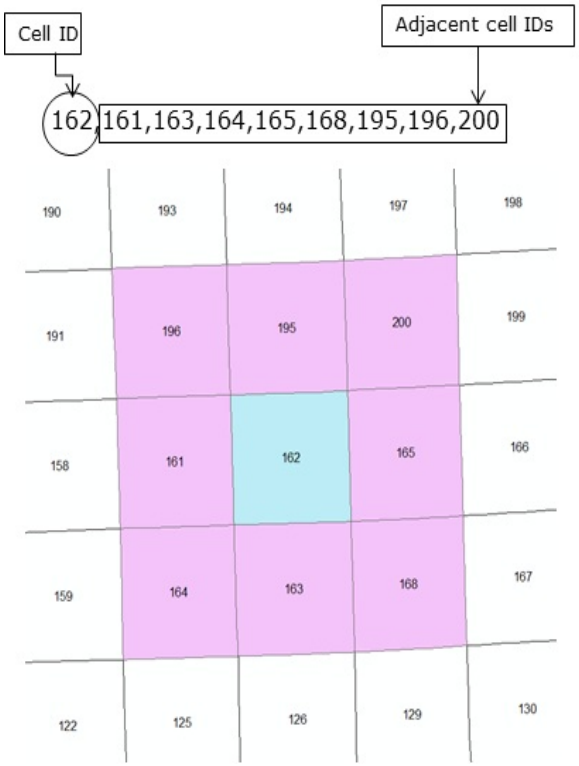

Figure 9.1: Adjacency matrix created in ArcMap using the Adjacency Toolbox.

9.6.2 Adjacency weights other than 1

If we don't want the adjacent 8 cells having equal weight, we can have weights based on neighbours that share common boundaries (Rook) or that share common boundaries and vertices (Queen). There are also distance-based matrices that can incorporate proximity, population densities, or covariates such as age or sex (Earnest et al. 2007) . Spatial resolution other than equal weight in the surrounding 8 cells can be handled and incorporated into modeling efforts using Conditional Autoregressive Models (CAR) in WinBUGS and can be created using the GeoDa program.

Earnest et al. (2007) identifies terms to describe several adjacency matrices that were based on neighborhood or distances that included:

1. *Queen* - neighborhood-based that refers to neighbors that share common boundaries and vertices (n=8 neighbors)

2. *Rook* - neighborhood-based that refers to neighbors that share common boundaries only (n=4 neighbors)

3. *Weights* - distance-based that refers to neighbors at various distances away are less influential

4. *Gravity* - distance-based that refers to neighbors that are more populated have greater influence

5. *Entropy* - distance-based that refers to neighbors that are closer provide more weight than those farther away

6. *Density* - distance-based similar to Gravity except refers to neighbors that have greater density and not just population size so takes into account area

7. *Covariate* - distance-based that identifies *a priori* knowledge of a variable as influential in determining a regions or cells disease rate

9.6.3 Covariates

We can extract covariates within each grid cell for any variable we have *a priori* knowledge that it may influence potential for transmission of TB. For example, the Michigan Department of Agriculture and Rural Development (MDA) provided georeferenced data and herd size (i.e., number of cattle per farm) for all farms in the 5 county area of the Modified Accredited Zone (MAZ) that encompassed about 8,074 km^2 of white-tailed deer habitat. We could have included a herd size effect in all models because these effects have been shown to influence *Mycobacterium bovis* (the bacteria responsible for TB) presence on farms or infection probability for farms in Europe (O'Reilly and Daborn 1995, Hutchings and Harris 1997, Phillips et al. 2003).

The main components of initiating a WinBUGS model section include:

1. Check Model

2. Load Data

3. Compiling chains

4. Load initial values

9.7 Check model

The Check Model component determines if the model structure is presented properly for the program to run. If the model structure is appropriate, the bottom left corner of the screen will read *model is syntactically correct* (Fig. 9.2).

```
model
{
for (i in 1 : NumFarms)
{
pos[i] ~ dbern(pi[i])
logit(pi[i]) <- mu[i]
mu[i] <- alpha + beta1*DeerDensity[i] + beta2*AP5yGrid[i] + beta3*PercWetland[i]
+ beta4*Sand[i] + beta5*SoilpH[i] + beta6*PreFreq[i] + b.car[cellid[i]] + epsi[cellid[i]]
}
```

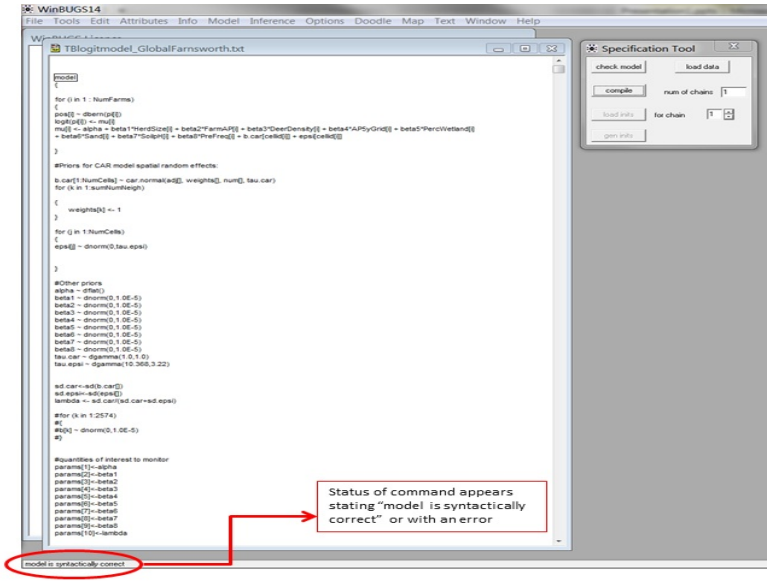

Figure 9.2: Compiling the model structure in WinBUGS.

9.8 Load data

The data needs to be separated by column and covariates reflect data for each positive and negative animal sampled. Example code is just an abbreviated version of data but each piece is separated by a comma after parenthesis for each variable. Note that the cell id is also necessary to include in the Load Data section. If the data is loaded successfully, the bottom left corner of the screen will read *data loaded* (Fig. 9.3).

```
list(NumFarms = 762, pos=c(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
cellid=c(24,31,23,31,31,23,23,31,31,31,24,24,31,35,31,24,31),
AP5yGrid=c(0,0,4.545454545,0,0,4.545454545,4.545454545,0)
```

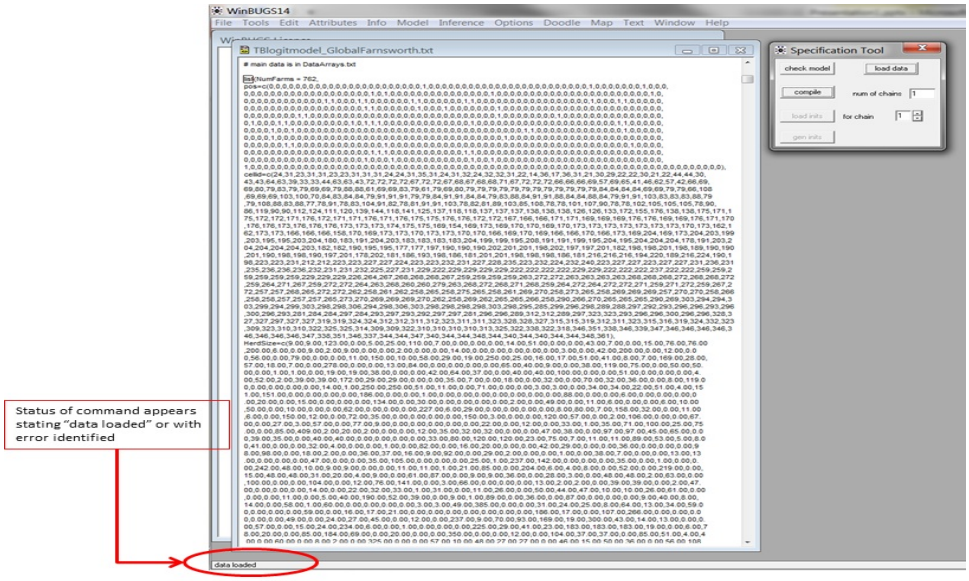

Figure 9.3: Loading the data into WinBUGS.

9.9 Compiling chains

Here you need to load the number of chains you plan to run before selecting the compile button. The number of chains will determine how many chains need to be loaded in the subsequent step. If this step is appropriate, the bottom left corner of the screen will read *model compiled* (Fig. 9.4).

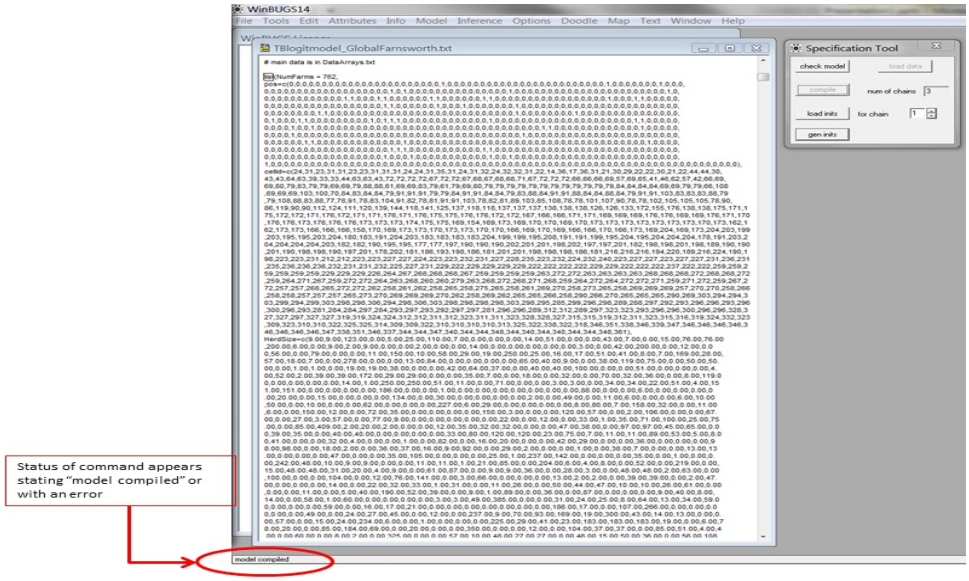

Figure 9.4: Compiling the number of chains in WinBUGS.

9.10 Load initial values

To load initial values after compiling the 3 chains, you need to select drag a box over the term *list* then seelct *load inits*. After doing this for each chain, you may have to select *gen inits* until the bottom left corner of the screen reads *initial values generated, model initialized* (Fig. 9.5).

```
#Initial values for Markov chains.
list(alpha = 0, beta1 = 0,  beta2 = 0, beta3 = 0, beta4 = 0, beta5 = 0, beta6 = 0,
     beta7 = 0, beta8 = 0)
list(alpha = 0, beta1 = 0,  beta2 = 0, beta3 = 0, beta4 = 0, beta5 = 0, beta6 = 0,
      beta7 = 0, beta8 = 0)
list(alpha = 0, beta1 = 0,  beta2 = 0, beta3 = 0, beta4 = 0, beta5 = 0, beta6 = 0,
      beta7 = 0, beta8 = 0)
```

9.11 Sample monitor tool

The *Sample Monitor Tool* allows WinBUGS to store every value it simulates for that parameter. This will enable us to view trace plots of the samples to check convergence and to obtain posterior quantiles for a parameter (Fig. 9.6). Parameters were set in the model statement below:

```
#quantities of interest to monitor
```

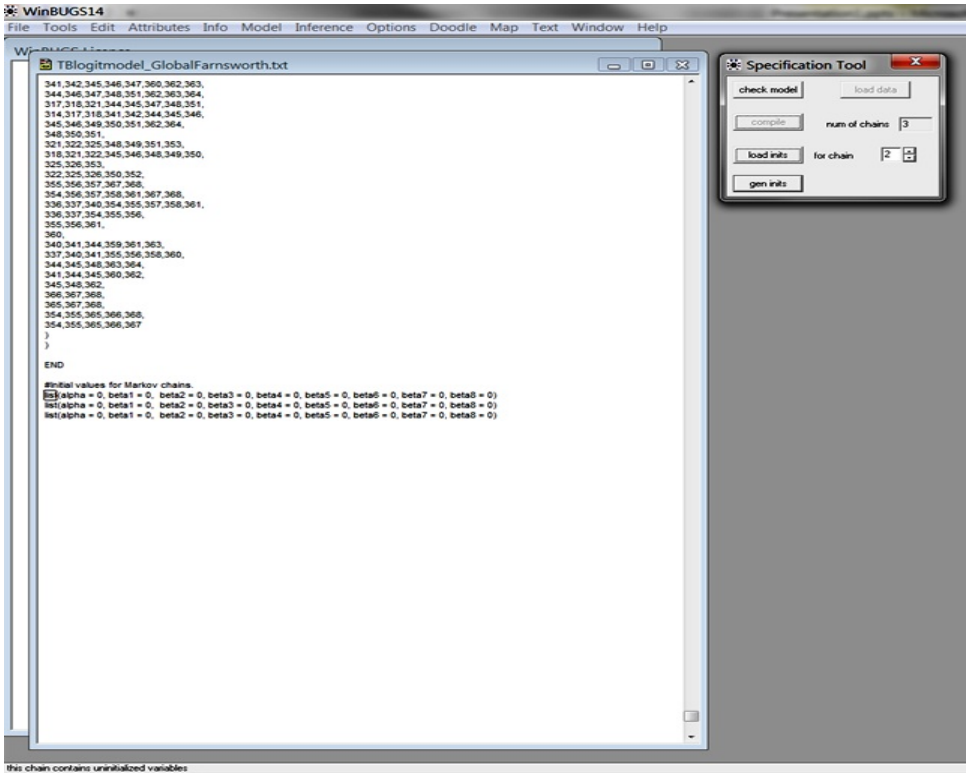

Figure 9.5: Loading the initial values for each chain in WinBUGS.

```
params[1]<-alpha
params[2]<-beta1
params[3]<-beta2
params[4]<-beta3
params[5]<-beta4
params[6]<-beta5
params[10]<-lambda
```

9.12 Update tool

The *Update Tool* under Model in WinBUGS is used to set the number of iterations, refresh to keep you up to date at what iteration the program is on, and the number to thin . This will enable us to view trace plots of the samples to check convergence and to obtain posterior quantiles for a parameter (Fig. 9.6).

9.13 Further considerations

1. *Prior Distributions* - Prior distributions (e.g., non-informative N (0, 100,000)) for each of the parameters, and over the entire real line for μ (e.g., an improper (flat) prior). Prior distributions for the random effect describing region-wide heterogeneity and to describe the spatial structure (e.g., intrinsic Gaussian conditional autoregressive (ICAR) prior with a sum to zero constraint) should also be determined. Because of the marginal specification for region-wide heterogeniety and spatial structure, prior distributions for

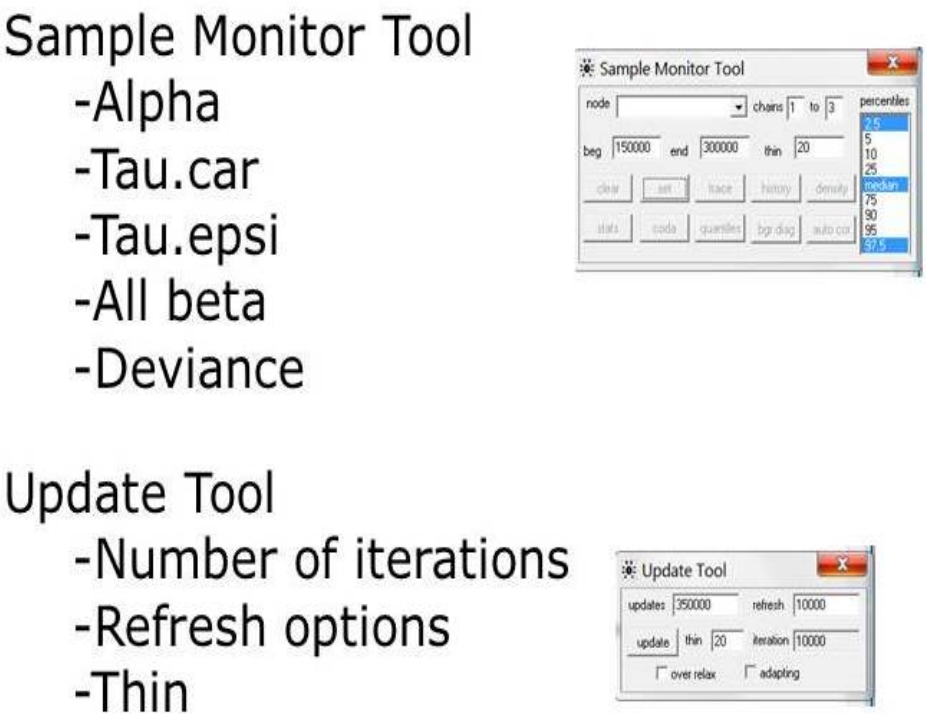

Figure 9.6: Setting the sample monitor tool and initiating program to run in WinBUGS.

the precisions using simulations in WinBUGS should be determined using the *psi* metric (Eberly and Carlin 2000).

2. *Model Selection* - candidate models can consist of different structures with strictly additive effects, environmental predictors can be grouped, such that they can all be entered or removed from the models together. Also, to account for the spatial structure of the data, random effects parameters can be included in some models to represent region-wide heterogeneity and local clustering. Therefore, all models can consist of all possible combinations of the grouped variables, other covariates, and random effects. Deviance information criterion (DIC) can then be used to evaluate this candidate set of models with DIC weights allowing for an intuitive comparison of the evidence in the data for each candidate model. The weights are considered a measure of the strength of evidence in the data for ith model being the "best" model of those within the candidate set, and therefore provide a measure of model selection uncertainty (Burnham and Anderson 2002, Spiegelhalter et al. 2002).

3. *Goodness-of-Fit* - to examine the goodness-of-fit of the top model from candidate sets, a numerical posterior predictive check can be conducted (Gelman et al. 2004). We can use the total number of positive subjects (farm's in our case) conditioned on the observed covariate values in our sample as our test statistic. Generating the posterior distribution of this statistic using parameter estimates from the marginal posterior distribution contained in MCMC chains. Thus given each farm's covariate values, we generated estimates of individual infection probabilities for every location for each of the 250,000 iterations of our MCMC chains, and created a Bernoulli random variable using this probability of M. bovis infection. We then summed these random variables across all farms to create our test statistic. The posterior distribution of the test statistic was created from the values of these test statistics across all iterations of our MCMC chains. Finally, we can calculate the posterior predictive P-value as the probability of having

fewer M. bovis-positive farms then the total number of infected farms observed in the sample based on this posterior distribution of the test statistic.

4. *Convergence and prior sensitivity* - examination of correlation and trace plots, as well as estimates of the corrected scale reduction factor for each parameter and multivariate potential scale reduction factors can provide evidence that chains for each model had converged. Additionally, the posterior distributions can be assessed to determine if they are overly sensitive to prior specification.