# Bayesian model in JAGS

## Tutorial 3 for transition

Zhe Zheng (Gigi)

2023-01-16

```r
knitr::opts_chunk$set(echo = TRUE)
require(rjags)
#"Note that the rjags package does not include a copy of
# the JAGS you must install this separately.
# For instructions on downloading JAGS,
# see the home page at https://mcmc-jags.sourceforge.io."
require(mcmcplots) # to plot the trace plots
require(coda) # for gelman diagnostic
```

# Outline

In this tutorial, we will learn how to use JAGS (Just Another Gibbs Sampler) in R to estimate unknown parameters with Markov Chain Monte Carlo (MCMC) methods in Bayesian framework. If you are interested in learning more about the theoretical knowledge of MCMC and Bayesian modeling, please check out courses "Probability and Statistics" (taught by Prof. Joseph Chang) and "Bayesian Statistics" (taught by Prof. Josh Warren).

In the first section, we will introduce a simple jags model to help understand the syntax for rjags package.

In the second section, we will learn how to build up complicated models step by step (censor data). Before reading this section, you will need to familiar yourself with the RSV_timing_explain tutorials and a research article on the relative RSV timing

In the third section, we will introduce the concept of spatial autocorrelation and how to incorporate this in jags model.

# Example 1: a simple jags model to estimate the unknown parameters

## First: Simulate a dataset for analysis

In this example, I simulate my own data in R in order to compare the MCMC results with our input parameters. I create a continuous outcome variable $y$ as a function of one predictor $x$ and an error term $\epsilon$ . I simulate 100 observations. The linear regression looks like

$$y = \beta_0 + \beta_1 x + \epsilon$$

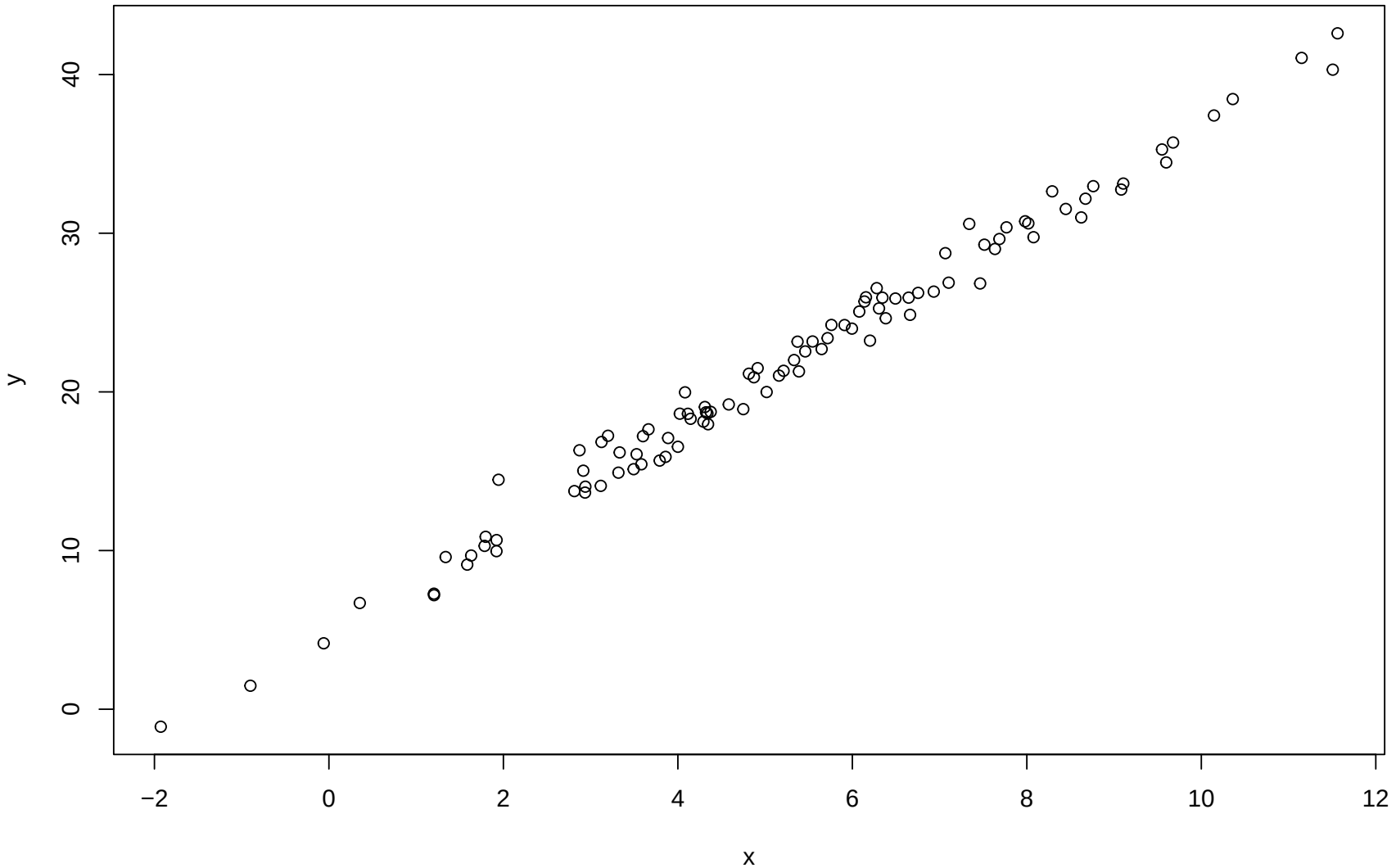
**Our goal is to estimate the unknown parameters $\beta_0$ and $\beta_1$ given the observations and the linear relationship**

```
n.sim=100; set.seed(123)
x=rnorm(n.sim, mean = 5, sd = 3)
epsilon=rnorm(n.sim, mean = 0, sd = 1)
beta0=5
beta1=3.2
y=beta0 + beta1 * x + epsilon
```

# Visualize the observations

Question: what is the intercept and slope for this linear regression?

```
plot(x,y,type="p")
```



# rjags mamual

The following is quoted from rjags document, authored by Martyn Plummer:

"JAGS is a clone of BUGS (Bayesian analysis Using Gibbs Sampling). See Lunn et al (2009) for a history of the BUGS project. Note that the rjags package does not include a copy of the JAGS library: you must install this separately. For instructions on downloading JAGS, see the home page at https://mcmc-jags.sourceforge.io.

To fully understand how JAGS works, you need to read the JAGS User Manual. The manual explains the basics of modelling with JAGS and shows the functions and distributions available in the dialect of the BUGS language used by JAGS. It also describes the command line interface. The rjags package does not use the command line interface but provides equivalent functionality using R functions."

# rjags mamual (continued)

"Analysis using the rjags package proceeds in steps: 1. Define the model using the BUGS language in a separate file.

2. Read in the model file using the jags.model function. This creates an object of class "jags".

3. Update the model using the update method for "jags" objects. This constitutes a 'burn-in' period.

4. Extract samples from the model object using the coda.samples function. This creates an object of class "mcmc.list" which can be used to summarize the posterior distribution. The coda package also provides convergence diagnostics to check that the output is valid for analysis (see Plummer et al 2006)."

# Jags model structure (Step 1)

```
basic_mod <-  "model{
 #model
  for(i in 1:n.sim){
   y[i] ~ dnorm(mu[i], tau)
# tau is the error term

   mu[i] = beta0 + beta1 * x[i]
# beta0 and beta1 are the unknown parameters
  }

 #priors; these are weakly informative priors
 beta0 ~ dnorm(0, 0.01)
# dnorm(mean, precision)
# precision = 0.01 --> var = 100
 beta1 ~ dnorm(0, 0.01)
 tau ~ dgamma(0.01,0.01)
# dgamma(shape,rate)
 }
 "
```

# Initialize model with known data and initial guess of unknown parameters (Step 2)

```
######## Define known data ############
datalist=list("y"=y,"x"=x,"n.sim"=n.sim)

######## Set initial values for parameter estimate ########
inits=function(){list("beta0"=rnorm(1), "beta1"=rnorm(1))}
```

# Read in the model file using the jags.model function (Step 3)

```
### Read in the model file using the jags.model function ##
jags.basic.mod = jags.model(textConnection(basic_mod),
                            data = datalist,
                            inits = inits,
                            n.chains = 4)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 100
##    Unobserved stochastic nodes: 3
##    Total graph size: 406
##
## Initializing model
```

# Discard burn-in period and sample for posterior distribution (Step 4)

```
############ burn-in period ############
update(jags.basic.mod,5000) # n.burnin = 1000

##### sample for posterior distribution #######
model_sample <- coda.samples(jags.basic.mod,
                             c("beta0","beta1"),
                             100000,thin=5)
#You can also use jags.samples(). The results will be the same
```

# Summarize the posterior distribution

```
summary(model_sample)
```

```
##
## Iterations = 5005:105000
## Thinning interval = 5
## Number of chains = 4
## Sample size per chain = 20000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##         Mean      SD  Naive SE Time-series SE
## beta0 4.983 0.21539 0.0007615      0.0010432
## beta1 3.183 0.03624 0.0001281      0.0001769
##
## 2. Quantiles for each variable:
##
##         2.5%   25%   50%   75% 97.5%
## beta0 4.560 4.839 4.984 5.128 5.405
## beta1 3.112 3.158 3.183 3.207 3.254
```

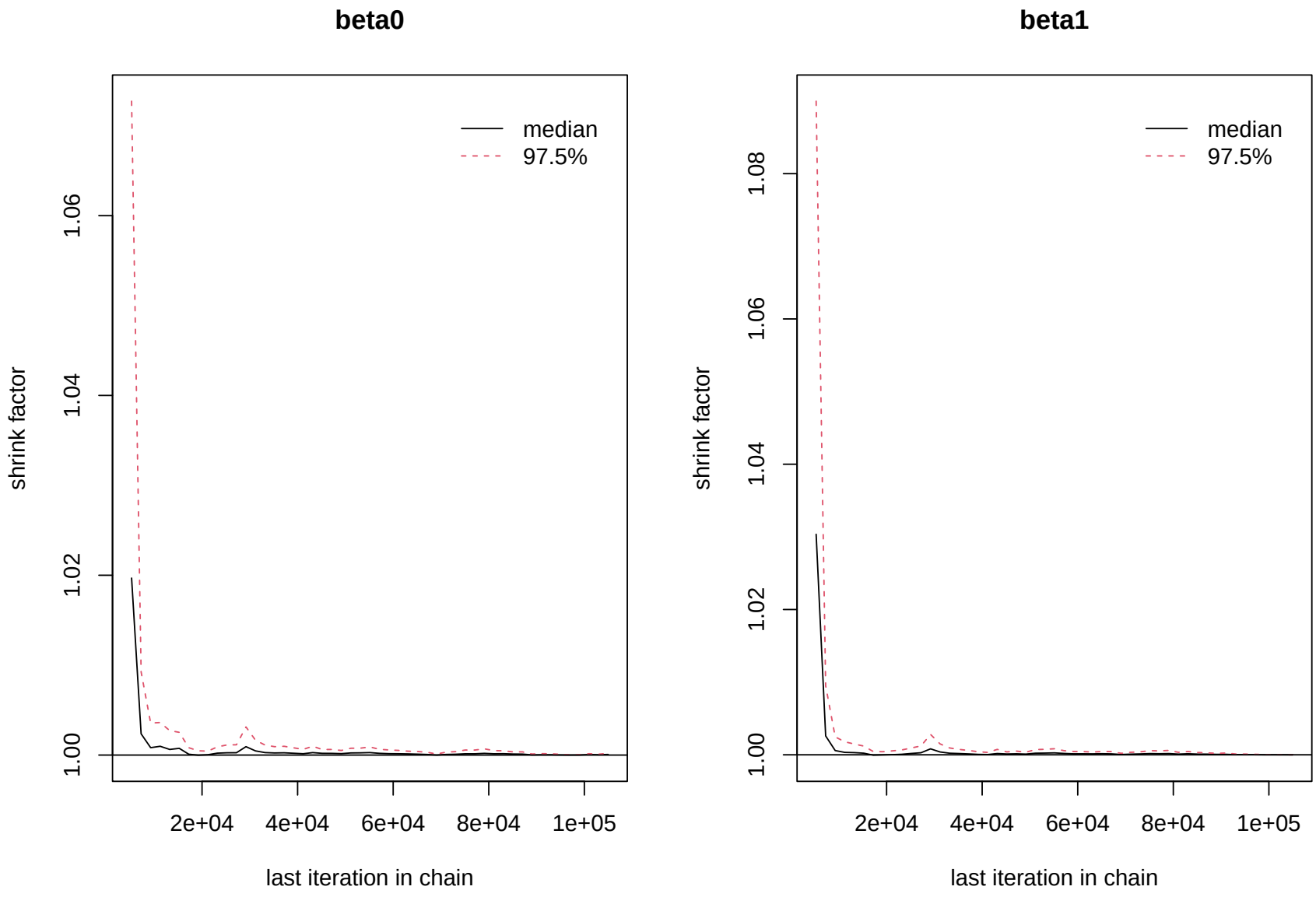
# Check for model convergence (gelman diagnotics)

```
gelman.diag(model_sample) ## <=1.1 means well converge
```

```
## Potential scale reduction factors:
##
##       Point est. Upper C.I.
## beta0          1          1
## beta1          1          1
##
## Multivariate psrf
##
## 1
```

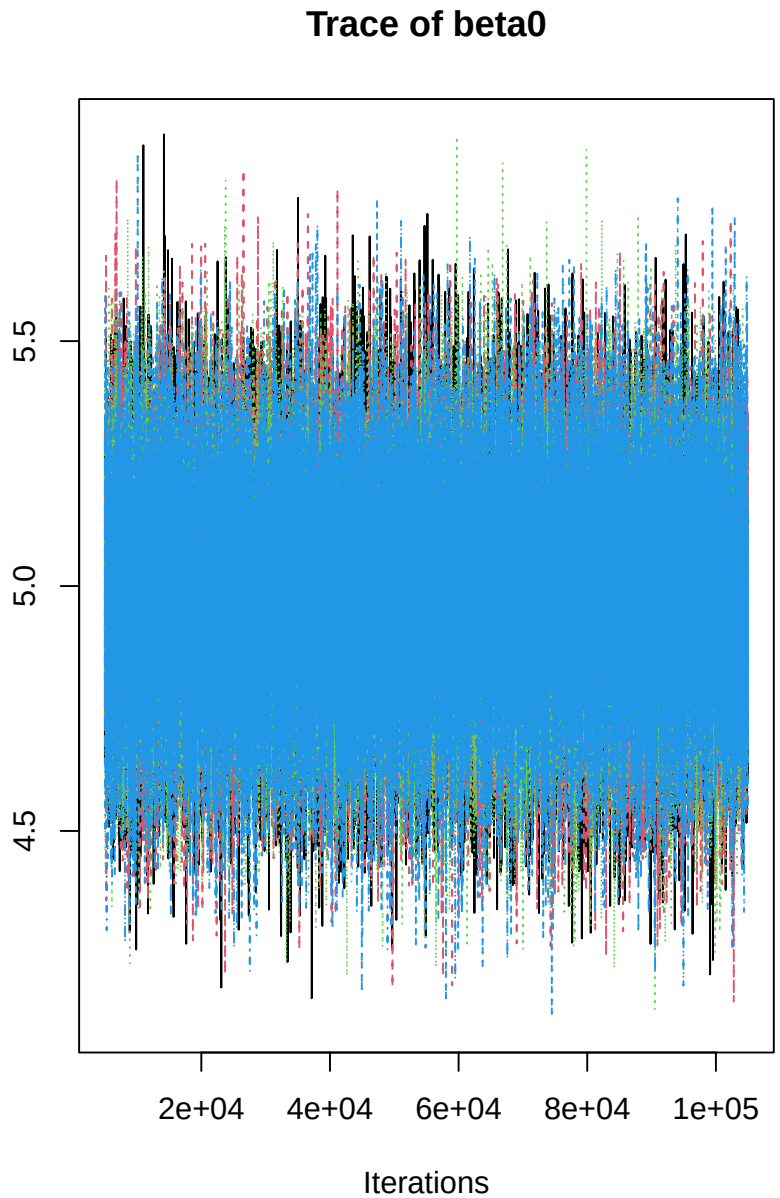
# Check for model convergence (gelman plot)

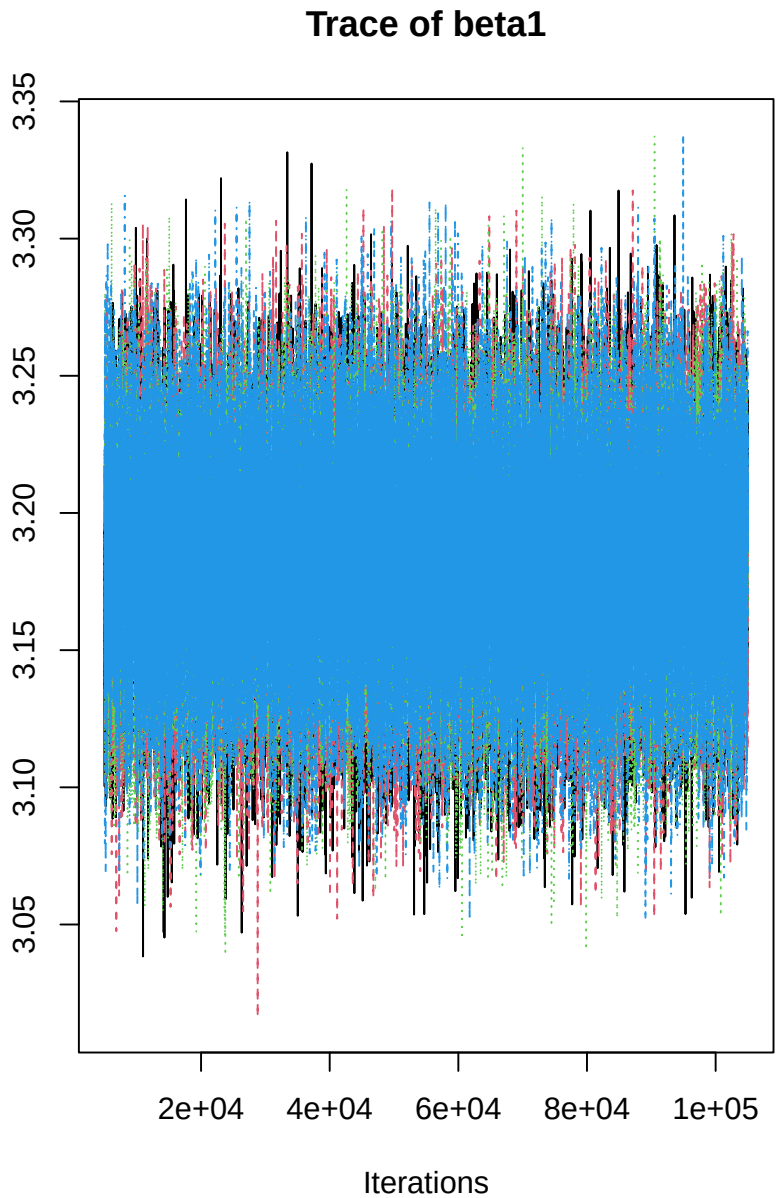
```
gelman.plot(model_sample)
```



# Plot the trace plot

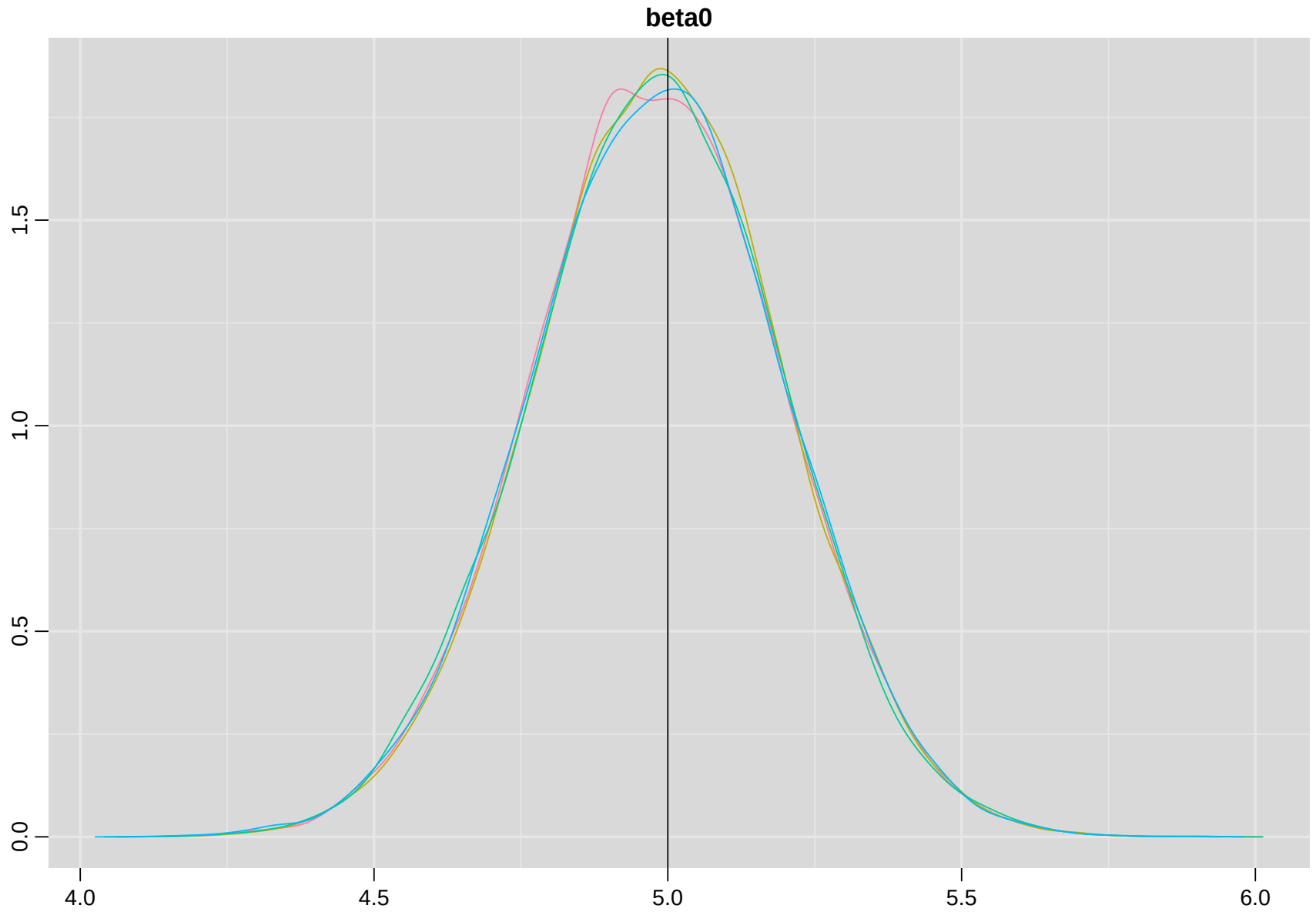
```
par(mfrow=c(1,2))
traceplot(model_sample)
```



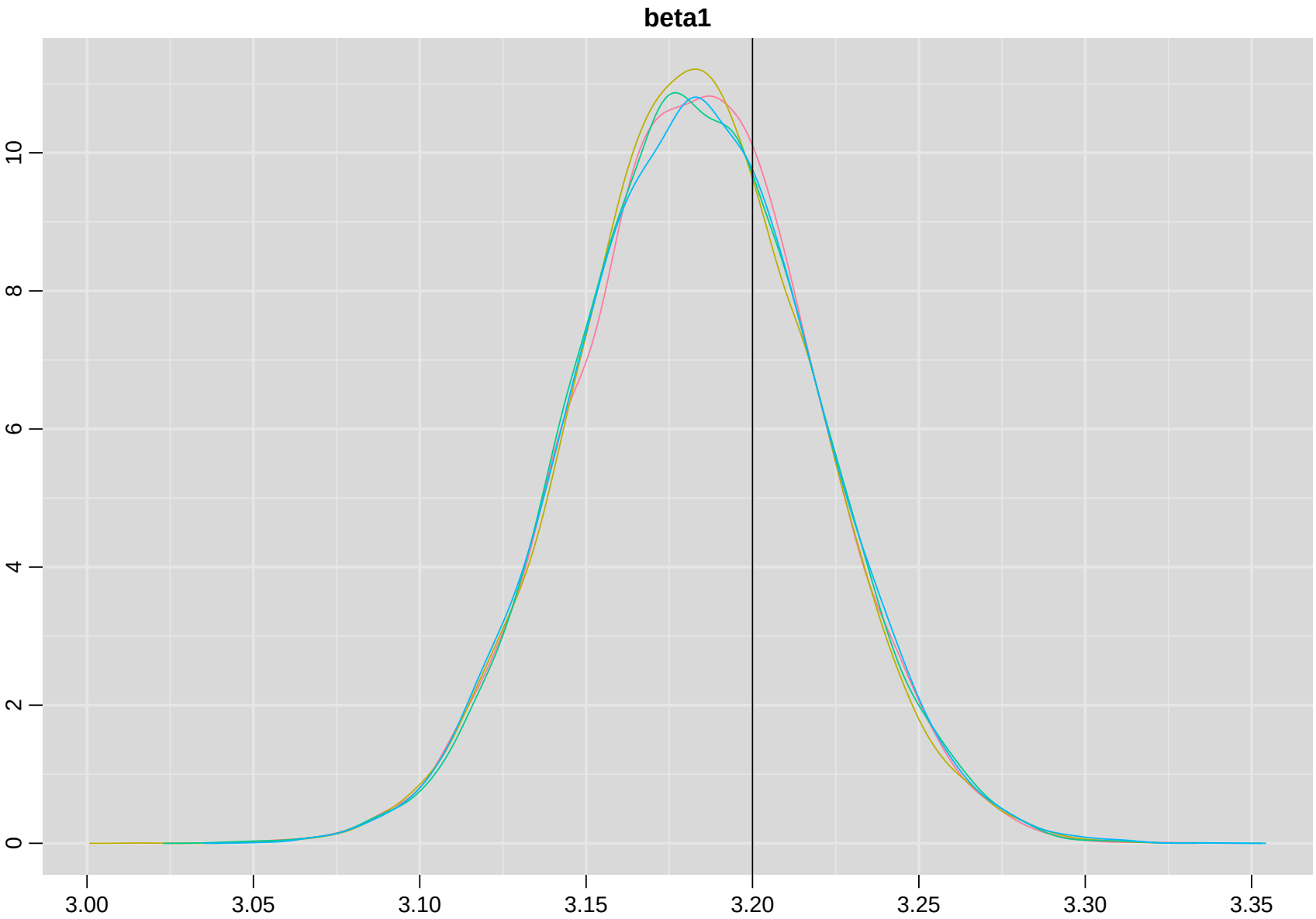
# Compare the posterior density and real value for beta0

```
denplot(model_sample, parms = c("beta0"))
abline(v=5)  # True value
```



# Compare the posterior density and real value for beta1

```
denplot(model_sample, parms = c("beta1"))
abline(v=3.2) # True value
```



# Section 2: estimate the parameters for the relative RSV timing

After we estimated the RSV timing using second derivative method (please check: RSV_timing_explain), we would like to know if some potential drivers contribute to the relative timing difference in RSV epidemic onsets before and during COVID-19 pandemic. (In reality, we manually coded the spatial autocorrelated model. As this requires taking Josh's classes, I will not go over it in the introduction tutorials.)

The model is given as:

$$Y_i = \alpha_{0i} + \beta_1 \overline{W_i} + \epsilon_i$$

$$\alpha_{0i} = \beta_0 + \boldsymbol{x_i^T \lambda} + \theta_i$$

# Meaning of the variables and parameters

where $Y_i$ is the relative timing of RSV epidemic onset with respect to Florida in 2021 in state $i$ (measured in weeks), and $\overline{w_i}$ is the average relative timing of RSV seasons in state $i$ compared to Florida from 2016-2019. The global intercept parameter $\beta_0$ and slope parameter $\beta_1$ describe the overall similarity between a typical year's relative onset timing across all states and the relative timings observed in 2021. The observation-level random effect $\epsilon_i$ $N(0, \sigma^2)$ accounts for residual variability in the data.

The state-specific intercepts, $\alpha_{0i} = \beta_0 + \boldsymbol{x_i^T \lambda} + \theta_i$, represent the state-level similarity in relative timing for 2021 compared to previous seasons, where $x_i$ is the vector of covariates that could potentially explain differences in the relative timing of RSV between 2021 and previous years. The covariates include population density, average household size, and stringency index of non-pharmaceutical interventions against COVID-19.

$\theta_i$ represents a spatially correlated random effect, which we will learn more in detail later.

# Jags model code without considering spatial autocorrelation

```
Nospatial_model <- "model{
################## model ###################################
for(i in 1:n){
Y[i] ~ dnorm(mu[i], inv_var)
# for the observed timing (we log transform Y already)
 mu[i] <- alpha_0[i] + beta1*log_W[i]
 # log_W is the log transform average relative timing of RSV

alpha_0[i]= beta0+x1[i]*lambda1+x2[i]*lambda2+lambda3*x3[i]
# this is alpha_0i = beta0 + X_i*lambda (coviariates)
}

################## priors ##################################
beta0~dnorm(0,0.0001)
beta1~dnorm(0,0.0001)
lambda1~dnorm(0,0.0001)
lambda2~dnorm(0,0.0001)
lambda3~dnorm(0,0.0001)
```

# Load the data and run the jags model to sample posteriors

```
## loading data from the working directory
data_nospatial <- readRDS("./Nospatial_data.rds")

### Read in the model file using the jags.model function ##
non_spatial <- jags.model(textConnection(Nospatial_model),
                          data=data_nospatial,n.chains = 4)
```

```
## Warning in jags.model(textConnection(Nospatial_model), data =
## Unused variable "Z" in data
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 0
##    Unobserved stochastic nodes: 53
##    Total graph size: 491
##
## Initializing model
```

```
############ burn-in period ############
update(non_spatial,10000)
```

# Summarize the posterior distribution

```
summary(non_spatial_sample)
```

```
##
## Iterations = 10020:110000
## Thinning interval = 20
## Number of chains = 4
## Sample size per chain = 5000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##             Mean      SD Naive SE Time-series SE
## beta0   -0.4700 100.417  0.71006        0.69274
## beta1   -0.4466 100.275  0.70905        0.69658
## inv_var  0.9652   9.081  0.06422        0.06624
## lambda1  0.7978 100.957  0.71387        0.71390
## lambda2  0.5631 100.059  0.70752        0.70010
## lambda3  0.3978 100.180  0.70838        0.70841
##
## 2. Quantiles for each variable:
##
```

# Check for model convergence (gelman diagnostic)

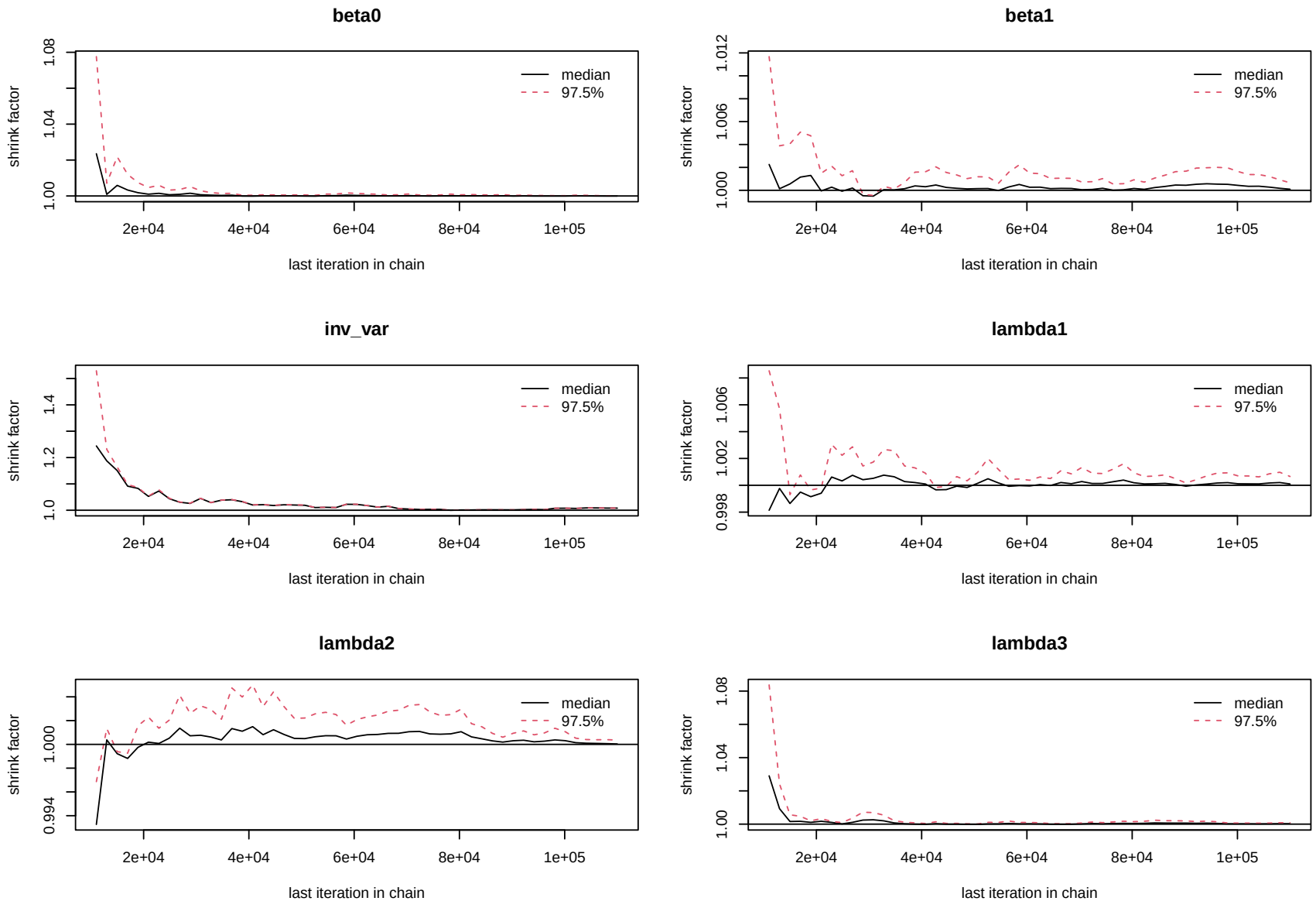
**inv_var did not converge well**. This indicates that we should either prolong the iterations or re-parameterize the model. Using gelman.plot (next slides), we could tell that prolong iterations may help convergence.

```
gelman.diag(non_spatial_sample)
```

```
## Potential scale reduction factors:
##
##          Point est. Upper C.I.
## beta0          1.00       1.00
## beta1          1.00       1.00
## inv_var        1.01       1.01
## lambda1        1.00       1.00
## lambda2        1.00       1.00
## lambda3        1.00       1.00
##
## Multivariate psrf
##
## 1
```

# Gelman plot

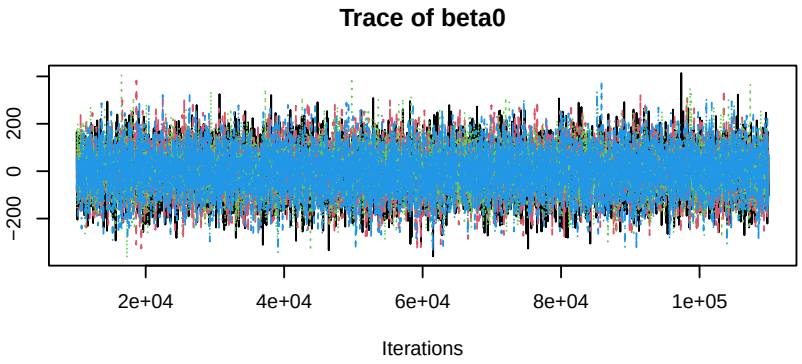
```
par(mfrow=c(3,2))
gelman.plot(non_spatial_sample)
```

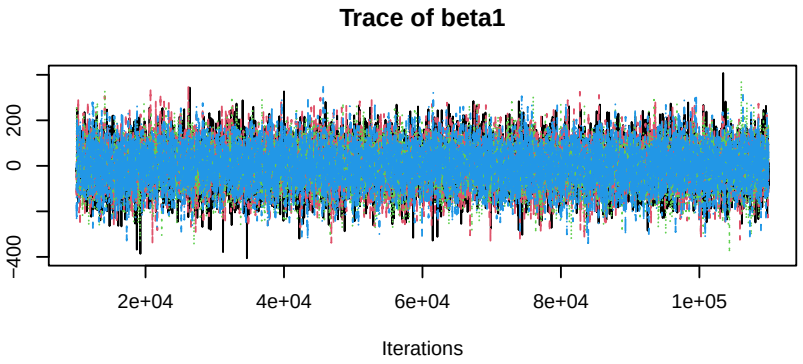


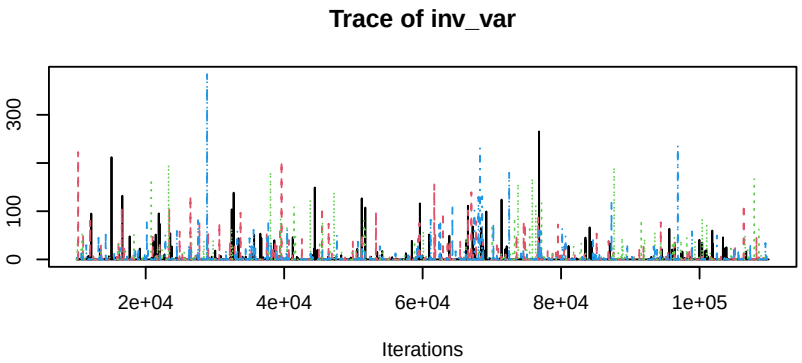
# Trace plots

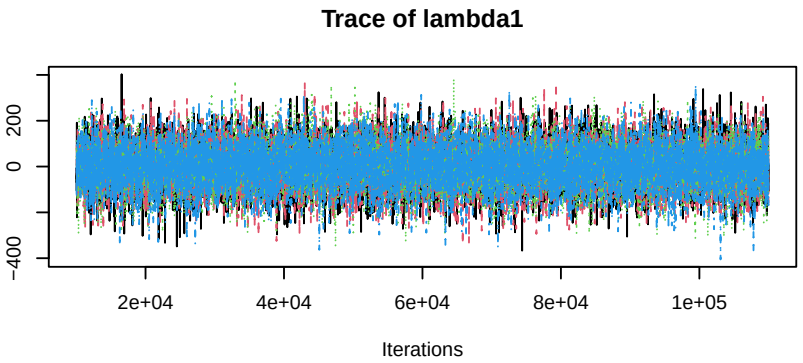
**note: inv_var is precision = the inverse of variance**

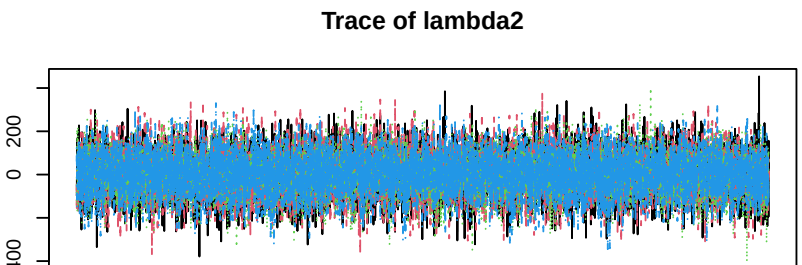
```
par(mfrow=c(3,2))
traceplot(non_spatial_sample)
```

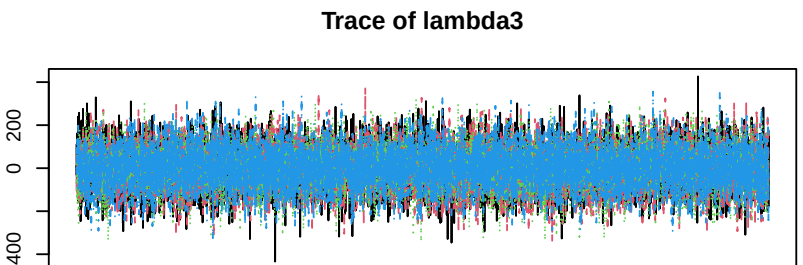


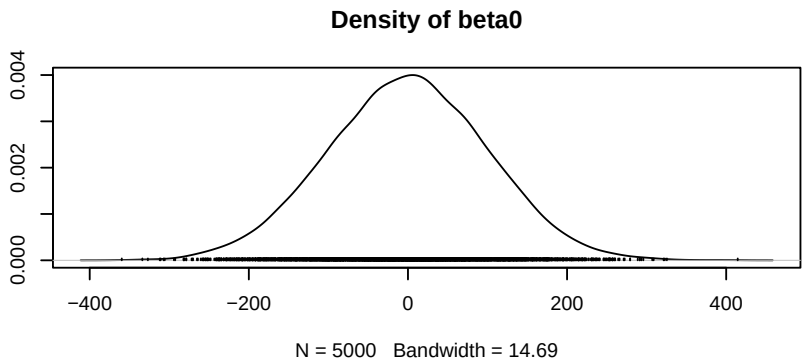



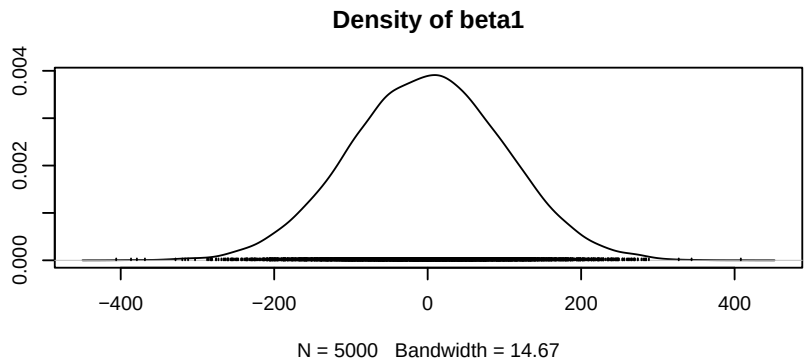



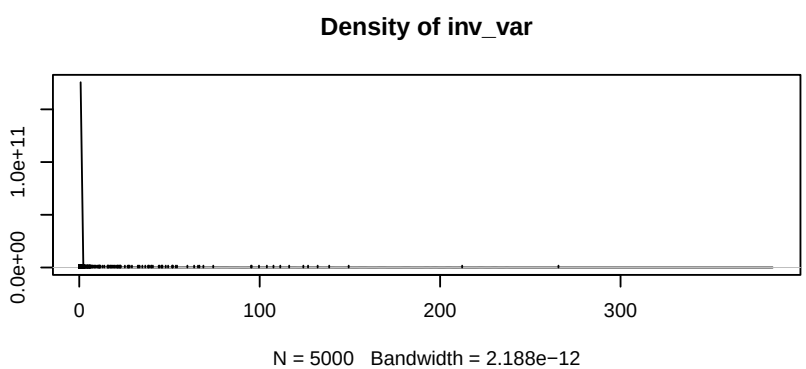



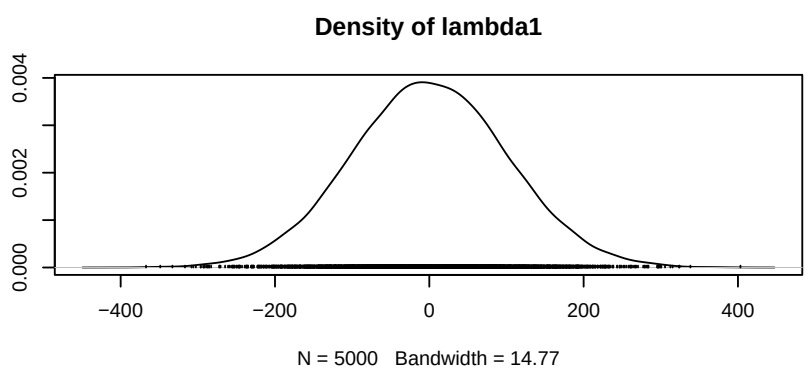
# Density plot of the posterior distributions

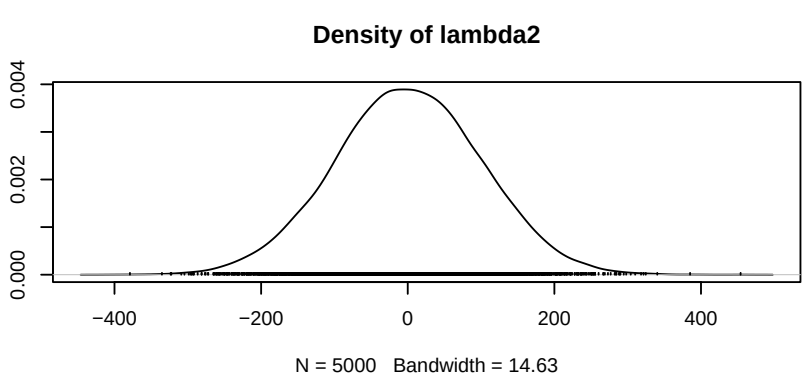
```
par(mfrow=c(3,2))
densplot(non_spatial_sample)
```

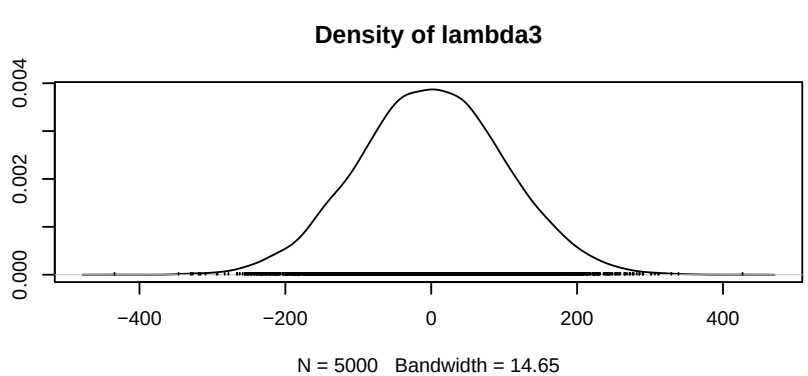


# Section 2.1: right censored survival analysis

In the early phase of an epidemic, we may not yet observe the onset point. For example, if you consider an onset threshold and the cases are still below the threshold, the epidemic has not taken off. In this case, we need to use right censored survival analysis.

Click to know more about right censored survival analysis

# Model structure for right censored survival analysis

```
censormodel <- "model{
for(i in 1:n){
is.censor[i] ~ dinterval(Z[i],t.cen)
# whether the data is censor and the censoring time

Z[i] ~ dlnorm(mu[i], inv_var)
# for the observed timing (we did not log transform Z)
# if we already log transform Z then we use dnorm

 mu[i] <- alpha_0[i] + beta1*log_W[i]
# log_W is the log transform average relative timing of RSV
alpha_0[i]= beta0+x1[i]*lambda1+x2[i]*lambda2
# this is alpha_0i = beta0 + X_i*lambda (coviariates)
}
```

# Model structure for right censored survival analysis (continued)

```
"model{# delete this one line of code
# when combine the two chunks. Separating the
# codes helps to show them in slides

################## priors ###################################
beta0~dnorm(0,0.0001)
beta1~dnorm(0,0.0001)
lambda1~dnorm(0,0.0001)
lambda2~dnorm(0,0.0001)
inv_var ~ dgamma(0.01, 0.01)}"
```

Since the model running process is the same, I will skip the model fitting process in this tutorial (it takes time to knit the PDF). To run the model, you need to combine the two chunks of model structures and change the eval from False to True.

# Same process to sample posteriors of right censored model

```
## loading data from the working directory
data_censor <- readRDS("./censor_data.rds")

### Read in the model file using the jags.model function ##
censor_model_jags <- jags.model(textConnection(censormodel),
                        data=data_censor,n.chains = 4)

############ burn-in period ############
update(censor_model_jags,10000)

######## sample for posterior distribution ##########
censor_model_sample <- coda.samples(censor_model_jags,
                                    c("beta0","beta1",
                                    "lambda1","lambda2",
                                    "lambda3","inv_var"),
                                     100000,thin=20)
```

# Section 3: spatial autocorrelation

Very often, the disease incidence is not randomly distributed across space. Observations in nearby localities tend to be more similar than observations further away, also known as positive spatial autocorrelation.

Prof. Duncan Lee has developed many statistical tools and tutorials to help evaluate spatial autocorrelation in public health. Please refer to the following materials to have an complete overview on Bayesian spatial model for public health and choosing conditional autoregressive priors:

- Bayesian Disease Mapping for Public Health
- Overview of Spatial Incidence Models
- CARBayes: An R Package for Bayesian Spatial Modeling with Conditional Autoregressive Priors
- Spatio-Temporal Areal Unit Modeling in R with Conditional Autoregressive Priors Using the CARBayesST Package

# A generalized linear mixed effect model (GLMM) framework

To produce accurate estimates of disease incidence, statistical models must take into account the full range of features present in the data, including non-normal distributions of count data, the impact of explanatory variables, and spatial correlation. One solution is to utilize generalized linear mixed models (GLMMs), which integrate a generalized linear model (GLM) with normally distributed random effects that account for both spatial correlation and overdispersion.

An example model structure is given as:

$$Y_i \sim Poisson(\mu_i)$$

$$\log \mu_i = \log E_i + x_i' \alpha + \phi_i$$

where $Y_i$ is the observed number of disease in region $i$, $\mu_i$ is the conditional mean, $E_i$ is the expected number of disease in region $i$, $x_i$ is the explanatory variables (covariates) of region $i$, $\alpha$ is the coeffiecients of the corresponding explanatory variables, and $\phi_i$ accounts for the random effects, which can be spatially correlated.

# Conditional autoregressive (CAR) priors for spatial modeling

To model spatial autocorrelation, four CAR priors are commonly used, including the intrinsic models, BYM models, Leroux models, and Cressie models. In **Duncan Lee's paper**, he systematically compared these four different priors and concluded that Leroux model is the best from both theoretical and practical standpoints. For short, Leroux model is appealing because it uses a single set of random effects to model the varying strengths of spatial autocorrelation $\rho$, where $\rho = 0$ means spatially independent while $\rho$ near 1 indicates strong spatial correlation.

# Leroux priors for spatial modeling

The Leroux version of spatial random effects is given as:

$$\phi_i|\boldsymbol{\phi_{-i}}, \rho, \tau^2 \sim MVN(\frac{\rho \sum_{j=1}^{n} w_{ij}\phi_i}{\rho \sum_{j=1}^{n} w_{ij} + 1 - \rho}, \frac{\tau^2}{\rho \sum_{j=1}^{n} w_{ij} + 1 - \rho})$$

where $\rho \in [0, 1)$ represents the spatial correlation that was estimated from the data; $\rho = 0$ corresponds to spatial independence, simplifying the model to an independent random effects model, and $\rho$ near 1 indicates strong spatial correlation. The $w_{ij}$ are variables that contain information about spatial proximity between two locations; $w_{ij}$ equals one if regions i and j share a common border and equal zero otherwise ($w_{ii}$=0 for all i). The total variance of spatial random effects is described by $\tau^2$.

# Create a spatial adjacency matrix

## Step 1. Set up – Shapefiles

(I learned this from Kayoko Shioda!)

```
# Load a shapefile. Make sure each polygon on shapefile
# corresponds to only one state.
require(rgdal) # for working with shapefiles
require(ggplot2) # for plot and fortify
CT1map = readOGR(dsn=path.expand('./WGS84'),
layer= 'zipct_37800_0000_2010_s100_census_1_shp_wgs84')
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "/Users/zhezheng/Library/CloudStorage/Box-Box/RSV_tut
## with 282 features
## It has 11 fields
## Integer64 fields read as strings:  ALAND10 AWATER10
```

```
CTmap.df <- as.data.frame(CT1map)
CT1map_fort <- fortify(CT1map,region="ZCTA5CE10")
# ZCTA5CE10 is CT zipcode ID
# The ID variable is then the zipcode/area unit
CT1map_fort$zipcode <- CT1map_fort$id
```

# How to create a spatial adjacency matrix

## Step 2. Find the number of neighbors from the sorted shapefile

```
# Sort both the shapefile and the data file by ID.
# This is the most important step
CTmap.sort <- CT1map[order(CT1map$ZCTA5CE10),]

library(spdep) # for neighbor evaluation
# Neighbors can either be Queen
# (any zip that touches another zip - even at corners)
# or Rook neighbors (share full sides -- not corners)

# The snap dist is governing the NA in neighbor matrix
# Can remove SNAP statement;
# if Queen = F then single point touch not allowed
# and Rook distance used.

# When data is missing -- there's more non-zero links involved
# Try to make snap distance as small as possible
# while making sure all zipcode have at least one neighbor
neighb <- poly2nb(CTmap.sort, queen = T, snap = sqrt(0.001))
```

# How to create a spatial adjacency matrix

## Step 3. Create a neighbor matrix with 1 suggesting adjacency

```
# Check average number of links -- increase snap distance
# if it says there is a zip with zero neighbors
neighb # Avg Number of links = 8.18705
```

```
## Neighbour list object:
## Number of regions: 282
## Number of nonzero links: 1518
## Percentage nonzero weights: 1.908858
## Average number of links: 5.382979
```

```
# Make neighbor matrix
# if zero.policy FALSE,
# stop with error for any empty neighbor sets,
# if TRUE, permit the weights list to be
# formed with zero-length weights vectors
# B is the basic binary coding,
# W is row standardised (both are commonly used)
neighb.mat <- nb2mat(neighb, zero.policy = T, style = 'B')
```

# How to create a spatial adjacency matrix

## Step 3. Create the W matrix for Leroux model

```
# For Leroux model, neighbor matrix needs to have 0
# if not neighboring, -1 if neighbor,
# and number of neighbors for each zip on diagonal), so...
w.mat <- ifelse(neighb.mat==1,-1,neighb.mat)
num.neigh <- colSums(neighb.mat)
# Count the number of neighbors for each state
for (i in 1:nrow(w.mat)) {
  # Put the number of neighbors on diagonal
  for (j in 1:ncol(w.mat)) {
    if (i==j) {
      w.mat[i,j] <- num.neigh[i]
    }
  }
}
```

# Spatial model with censored data

```
censor_spatial_model <- "model{
for(i in 1:n){
is.censor[i] ~ dinterval(Z[i],t.cen)
Z[i] ~ dlnorm(mu[i], inv_var)
 mu[i] <- beta0[i] + beta1*log_W[i]
}

beta0[1:n] ~ dmnorm(intercepts, Leroux)
 for(i in 1:n){
 intercepts[i] <- b0+x1[i]*b1+x2[i]*b2}

beta1~dnorm(0,0.0001)

b0~dnorm(0,0.0001)
b1~dnorm(0,0.0001)
b2~dnorm(0,0.0001)
```

# Spatial model with censored data (continued)

```
"model{# delete this one line of code
# when combine the two chunks. Separating the
# codes helps to show them in slides

Leroux <- inv.tau*(rho*w.mat + (1 - rho)*Id.mat)
# calculate spatial correlated random effects
  inv.tau ~ dgamma(0.01, 0.01)
# note the w.mat here is equal to
# diag(rowSums(neighbors_mat)) - neighbors_mat

  rho~ dunif(0,1)
inv_var ~ dgamma(0.01, 0.01)}"
```