Esercitazione 5

November 7, 2024

1 Modelli Bayesiani

1.1 Dugonghi

I Dudonghi di mare (meglio noti come Dugonghi) sono grandi mammiferi marini erbivori appartenenti alla famiglia Dugongidae, e strettamente imparentati con i lamantini. Si trovano principalmente nelle acque costiere poco profonde e nelle lagune tropicali dell'Oceano Indiano e del Pacifico occidentale, come le coste dell'Australia, dell'Africa orientale, dell'India e delle isole del Sud-est asiatico.

I dugonghi si nutrono di fanerogame marine (piante marine come le Zosterae e Halodule) e sono per questo motivo noti anche come "mucche di mare." Sono creature lente e pacifiche, capaci di vivere per molti anni e che, per via della loro dieta e dell'abitudine di vivere in aree marine vicine alla costa, sono vulnerabili alle attività umane, come la pesca e la distruzione degli habitat costieri.

1.2 I dati

Nel dataset, riportato sotto, avete la lunghezza di alcuni dudonghi Y_i e la loro età x_i.

[ ]:
```
## Your existing R code goes here
# library(jpeg)
# library(png)
# d4 <- readJPEG("d4.jpg") # Use readPNG() for PNG images
# d1 <- readPNG("/Users/gianlucamastrantonio/Dropbox (Politecnico di Torino␣
 ↪Staff)/Didattica/statistica computazionale/esercizi/d1.png") # Use readPNG()␣
 ↪for PNG images
# plot(c(0,1), c(0,0.5), type = "n", xlab = "", ylab = "", xaxt = "n", yaxt =␣
 ↪"n", bty = "n")

# rasterImage(d1, 0, 0, 1/2, 1/2)
# rasterImage(d4, 1 / 2, 0, 2 / 2, 1 / 2)
```

1.3 I dati

Nel dataset, riportato sotto, avete la lunghezza di alcuni dudonghi Y_i e la loro età x_i.

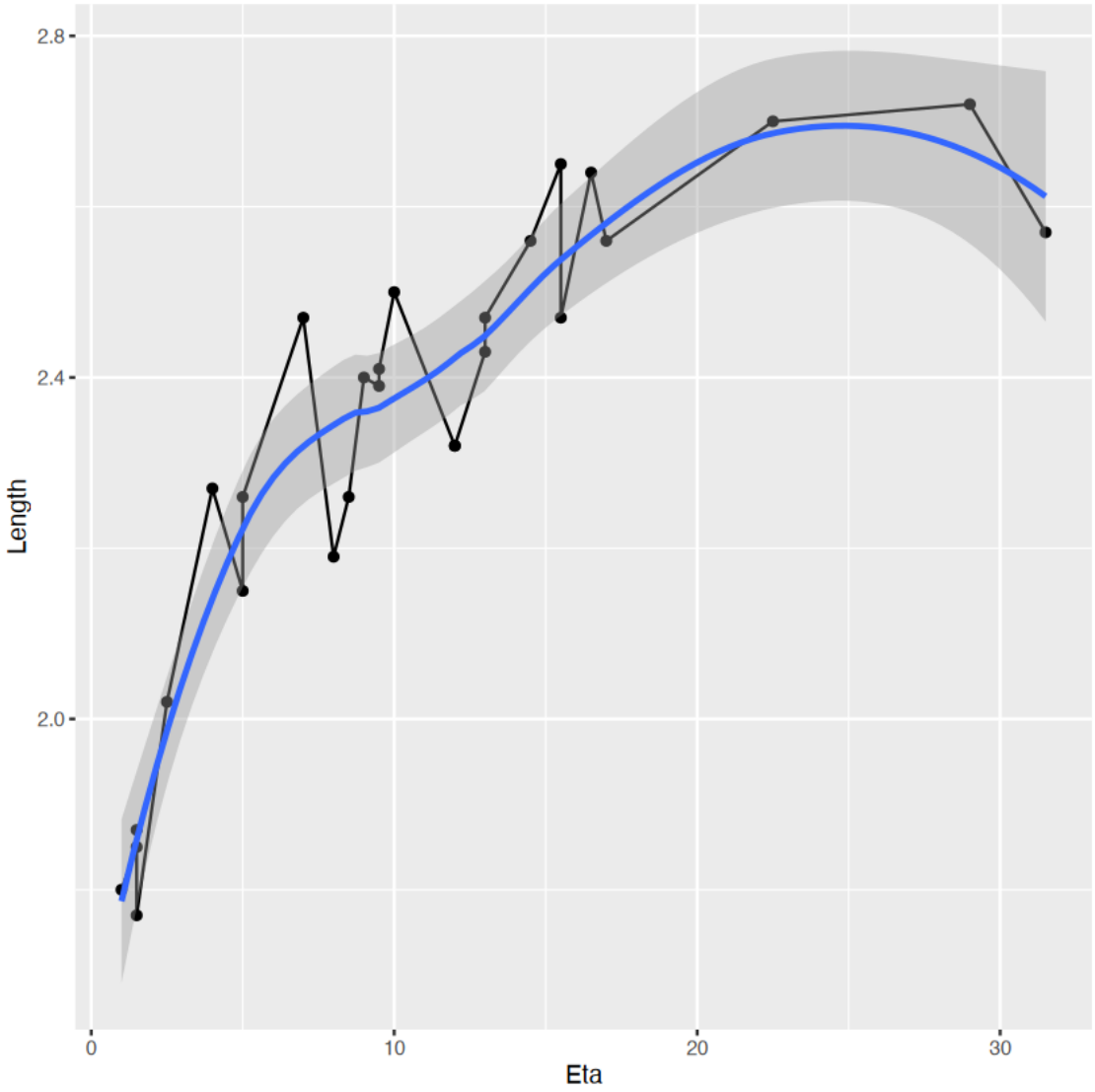
[2]:
```
# valori x
set.seed(1234)
x <- c(
  1.0, 1.5, 1.5, 1.5, 2.5, 4.0, 5.0, 5.0, 7.0,
  8.0, 8.5, 9.0, 9.5, 9.5, 10.0, 12.0, 12.0, 13.0,
  13.0, 14.5, 15.5, 15.5, 16.5, 17.0, 22.5, 29.0, 31.5
)
# valori y
Y <- c(
```

```
  1.80, 1.85, 1.87, 1.77, 2.02, 2.27, 2.15, 2.26, 2.47,
  2.19, 2.26, 2.40, 2.39, 2.41, 2.50, 2.32, 2.32, 2.43,
  2.47, 2.56, 2.65, 2.47, 2.64, 2.56, 2.70, 2.72, 2.57
)
```

Vediamo un semplice plot

[3]:
```
library(tidyverse)
library(ggplot2)
library(magrittr)

data_plot <- data.frame(Length = Y, Eta = x)
p <- data_plot %>%  ggplot(aes(x = Eta, y = Length)) + geom_line() + ␣
 ↪geom_point() + geom_smooth()
p
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse
2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.1
v purrr     1.0.2
-- Conflicts ------------------------------------------
tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package
(<http://conflicted.r-lib.org/>) to force all conflicts to
become errors

Caricamento pacchetto: 'magrittr'


Il seguente oggetto `e mascherato da 'package:purrr':

    set_names


Il seguente oggetto `e mascherato da 'package:tidyr':

    extract


`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



come ci aspettiamo la cresciata non è lineare. Per i dati assumuiamo due verosimiglianze diverse. Il modello 1 (**M1**) è

$$Y_i \overset{iid}{\sim} N(\mu_i, \sigma^2)$$

Il modello 2 (**M2**) è

$$Y_i \overset{iid}{\sim} G(a_i, b)$$

Nel secondo modello abbiamo che

$$E(Y_i) = \mu_i = \frac{a_i}{b}$$

e

$$Var(Y_i) = \sigma_i^2 = \frac{a_i}{b_i^2} = \frac{\mu}{b_i}$$

con $\sigma^2 = \frac{1}{b}$ Quindi

$$b = \frac{1}{\sigma^2}$$

e

$$a_i = \frac{\mu_i}{\sigma^2}$$

Domande

1. Scrivete una funzione per un modello Bayesiano con $\mu_i = \beta_0 + \beta_1 x_i$, ottenete campioni a posteriori, e mostrate graficamente la media a posteriori di μ_i e il suo intervallo di credibilità per ogni valore di x. Come prior utilizzate $\beta_j \sim N(0, \tau^2)$ e $\sigma^2 \sim IG(a, b)$.
2. Assumete che la media sia $\beta_0 - \beta_1 \gamma^{x_i}$ con $\gamma \in [0, 1], \beta_0 > 0, \beta_2 < \beta_1$ e implementato un modello per stimare la curva di crescita e la sua distribuzione, con approriate prior
3. Rifate il punto 2 assumendo M2 come verosimiglianza, e proponete tutti i parametri con una Normale (univariata o Multivariata), adattando la varianza. Confrontate i risultati, le curve di crescita, e utilizzando qualche metrica (CRPS, MSE, AIC) decidete quale dei due è migliore

1.3.1 Soluzioni

1. Scrivete una funzione per un modello Bayesiano con $\mu_i = \beta_0 + \beta_1 x_i$, ottenete campioni a posteriori, e mostrate graficamente la media a posteriori di μ_i e il suo intervallo di credibilità per ogni valore di x. Come prior utilizzate $\beta_j \sim N(0, \tau^2)$ e $\sigma^2 \sim IG(a, b)$.

[4]:
```
## Punto 1
#library(MCMCpack) # per la distribuzione Inverse Gamma
library(MASS) # per la distribuzione multivariata normale

bayesian_regression <- function(Y, X, tau2 = 10, a = 2, b = 1, n_iter = 10000) {

  # Inizializzazione
  beta_samples <- matrix(NA, n_iter, ncol(X)) # Per salvare i campioni di beta
  sigma2_samples <- numeric(n_iter) # Per salvare i campioni di sigma^2

  # Valori iniziali
  beta <- rep(0, ncol(X))
  sigma2 <- 1

  # MCMC
  for (i in 1:n_iter) {
    # Aggiorna beta condizionatamente a sigma^2 e Y
    V_beta <- solve(t(X) %*% X / sigma2 + diag(1 / tau2, ncol(X)))
    m_beta <- V_beta %*% t(X) %*% Y / sigma2
    beta <- mvrnorm(1, m_beta, V_beta)

    # Aggiorna sigma^2 condizionatamente a beta e Y
    residuals <- Y - X %*% beta
    shape <- a + length(Y) / 2
    rate <- b + sum(residuals^2) / 2
    sigma2 <- 1/rgamma(1, shape = shape, rate = rate)

    # Salva i campioni
```

```
    beta_samples[i, ] <- beta
    sigma2_samples[i] <- sigma2
  }

  # Risultati
  list(
    beta_samples = beta_samples,
    sigma2_samples = sigma2_samples
  )
}
```

```
Caricamento pacchetto: 'MASS'


Il seguente oggetto `e mascherato da 'package:dplyr':

    select


Il seguente oggetto `e mascherato da 'package:dplyr':

    select
```

Intuitivamente so decidere dei valori per la prior di β_j. Per quella di σ^2 è più complicato, e o si usano valori standard (a=b=1), che non sempre sono giusti, oppure la potete plottare e decidere.

[5]:
```
library(MCMCpack)
data_plot_ig <- data.frame(xseq = seq(0.0001, 3, by = 0.01), "a=1 b=1" = 0,␣
 ↪"a=2 b=1" = 0, "a=2 b=2" = 0, "a=1 b=2" = 0, "a=0.1 b=1" = 0, "a=0.1 b=0.1"␣
 ↪= 0, "a=1 b=0.1" = 0)
data_plot_ig[, "a=1 b=1"] <- dinvgamma(data_plot_ig$xseq, 1, 1)
data_plot_ig[, "a=2 b=1"] <- dinvgamma(data_plot_ig$xseq, 2,1)
data_plot_ig[, "a=2 b=2"] <- dinvgamma(data_plot_ig$xseq, 2, 2)
data_plot_ig[, "a=1 b=2"] <- dinvgamma(data_plot_ig$xseq, 1, 2)
data_plot_ig[, "a=0.1 b=1"] <- dinvgamma(data_plot_ig$xseq, 0.1, 1)
data_plot_ig[, "a=0.1 b=0.1"] <- dinvgamma(data_plot_ig$xseq, 0.1, 0.1)
data_plot_ig[, "a=1 b=0.1"] <- dinvgamma(data_plot_ig$xseq, 1, 0.1)
```

```
Caricamento del pacchetto richiesto: coda

##
## Markov Chain Monte Carlo Package (MCMCpack)

## Copyright (C) 2003-2024 Andrew D. Martin, Kevin M. Quinn, and Jong Hee Park

##
```

```
## Support provided by the U.S. National Science Foundation

## (Grants SES-0350646 and SES-0350613)
##
```

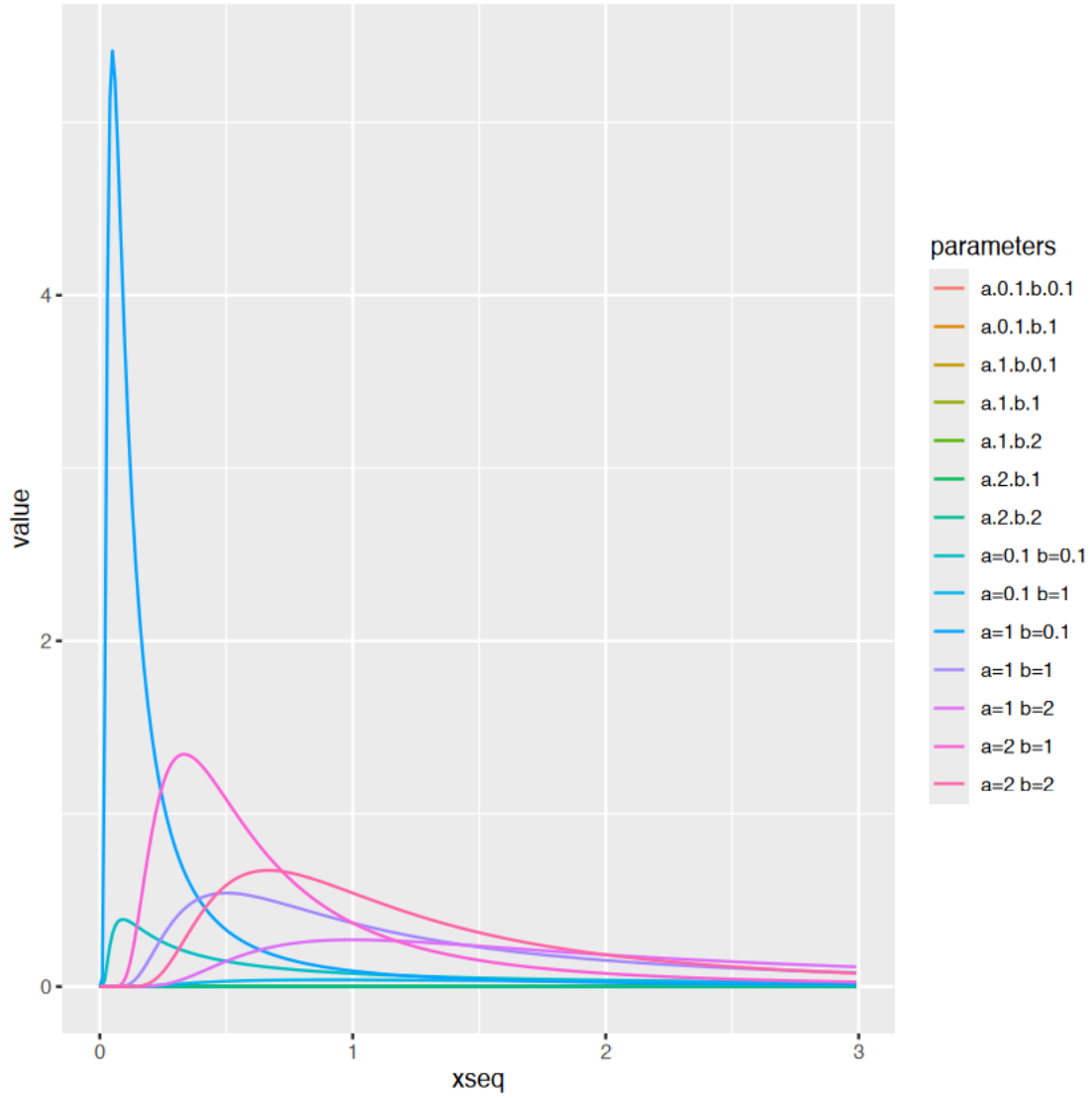
Potrei plottarli con le classiche funzioni di R, oppure usare ggplot. Per ggplot dovremmo avere le variabili di interessa in una sola colonna, e poi avere una variabile che ci indica ogni riga a cosa corrisponde. Si può trasfomrare il dataset facilmente con tidyverse

[6]:
```
data_plot_ig_long <- data_plot_ig %>% pivot_longer(
  cols = colnames(data_plot_ig)[-1],
  names_to = "parameters",
  values_to = "value"
)
data_plot_ig_long[1:10,]
```

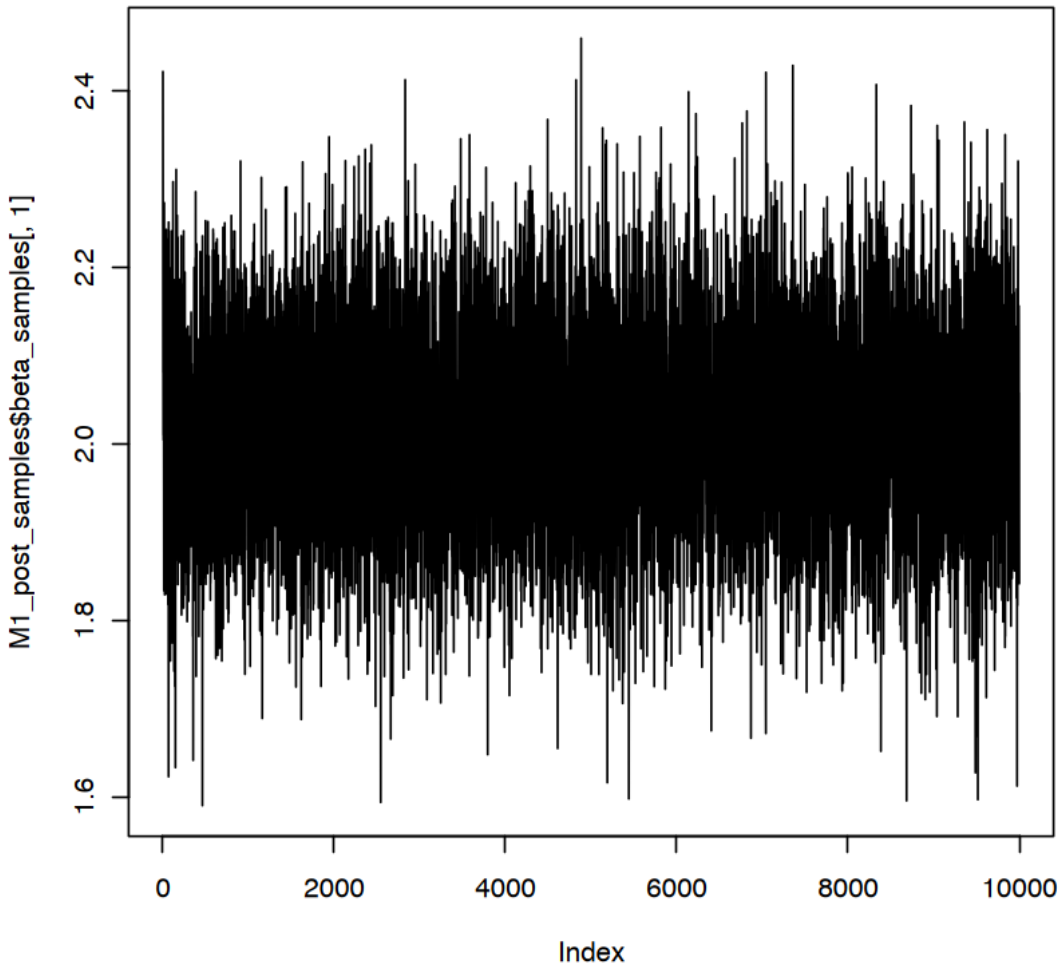
A tibble: 10 x 3

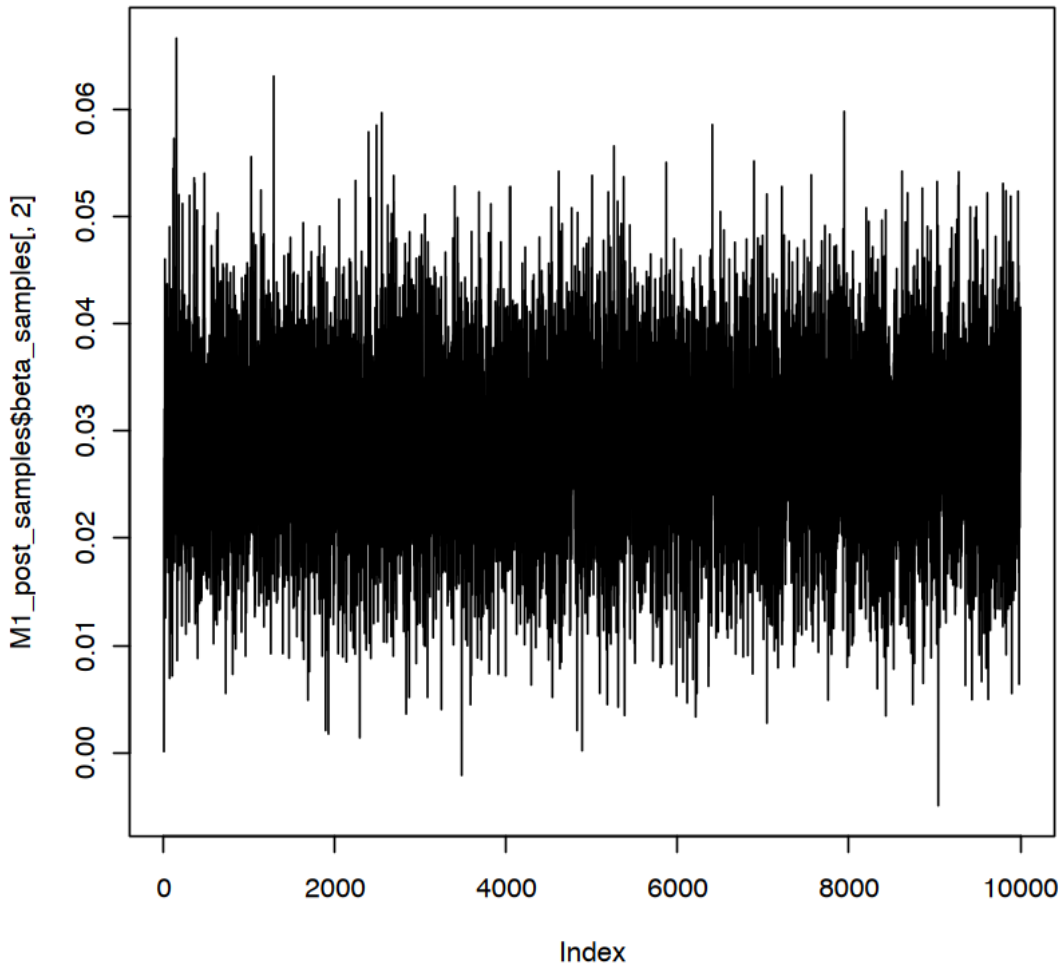
xseq <dbl>	parameters <chr>	value <dbl>
1e-04	a.1.b.1	0
1e-04	a.2.b.1	0
1e-04	a.2.b.2	0
1e-04	a.1.b.2	0
1e-04	a.0.1.b.1	0
1e-04	a.0.1.b.0.1	0
1e-04	a.1.b.0.1	0
1e-04	a=1 b=1	0
1e-04	a=2 b=1	0
1e-04	a=2 b=2	0

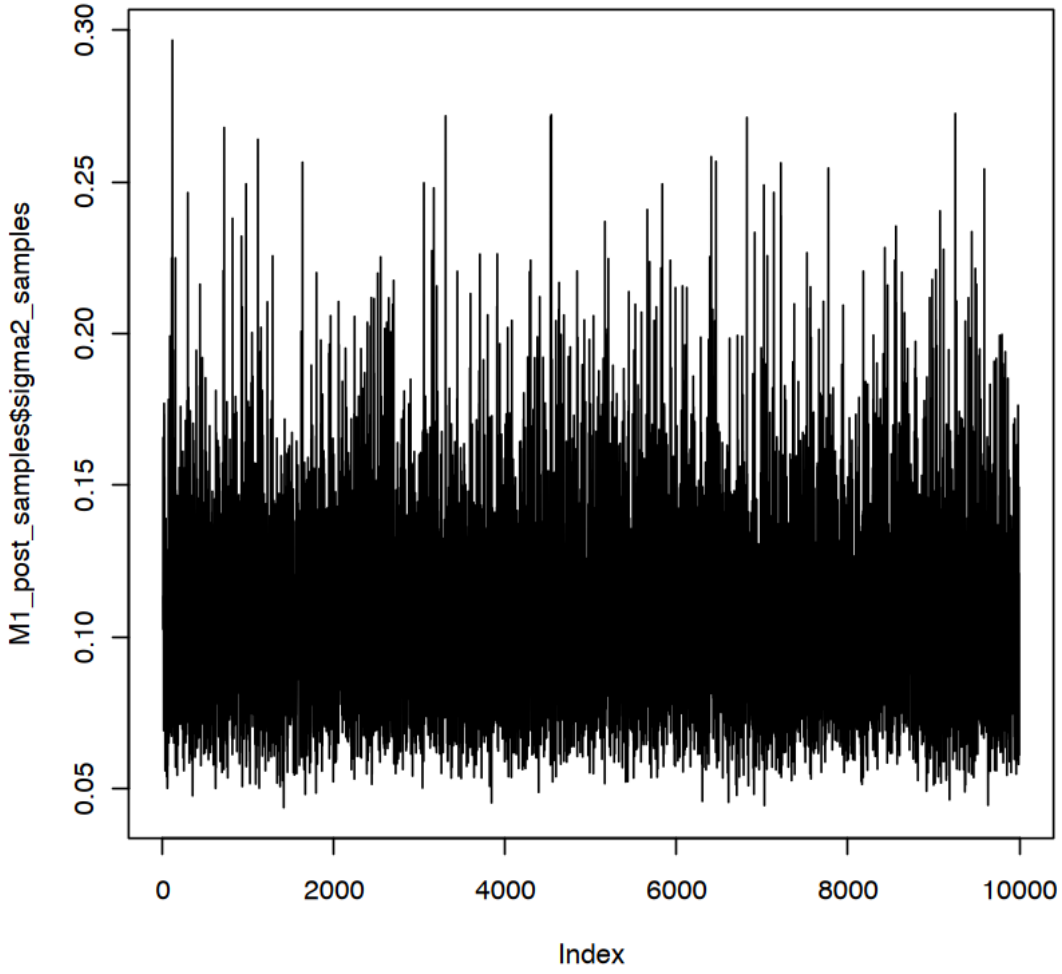
[7]:
```
data_plot_ig_long %>% ggplot(aes(x = xseq, y = value,  col= parameters)) +␣
 ↪geom_line()
# il problema è che li
```



[8]:
```
n <- length(Y)
M1_post_samples <- bayesian_regression(Y = Y, X = cbind(rep(1,n), x), tau2 =␣
 ↪10000, a = 1, b = 1, n_iter = 10000)
# e vediamo le catene
plot(M1_post_samples$beta_samples[,1], type="l")
plot(M1_post_samples$beta_samples[, 2], type="l")
plot(M1_post_samples$sigma2_samples, type = "l")
```


M1_post_samples$beta_samples[, 1]
1.6
1.8
2.0
2.2
2.4
0
2000
4000
6000
8000
10000
Index

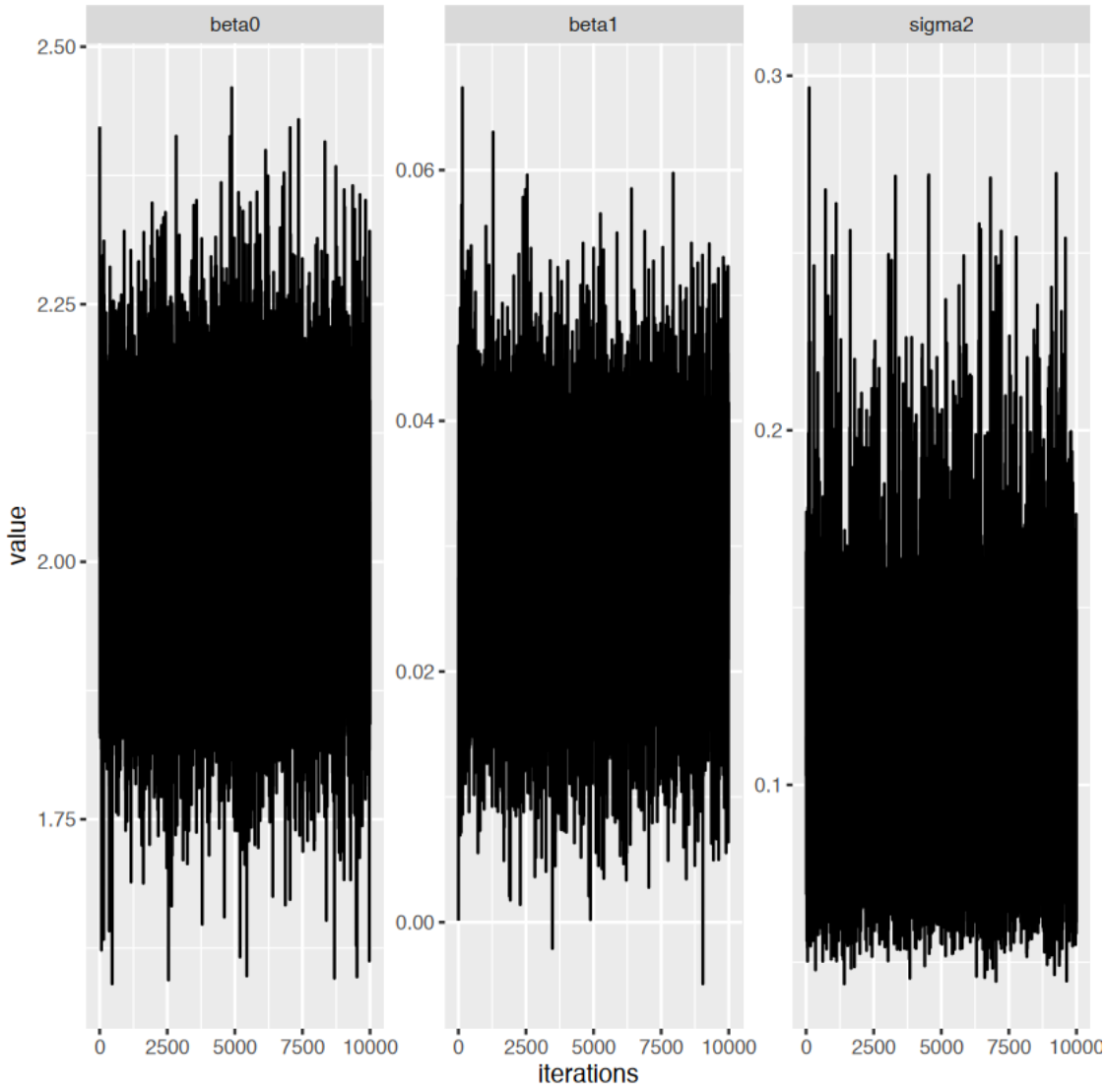

M1_post_samples$beta_samples[, 2]
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0
2000
4000
6000
8000
10000
Index

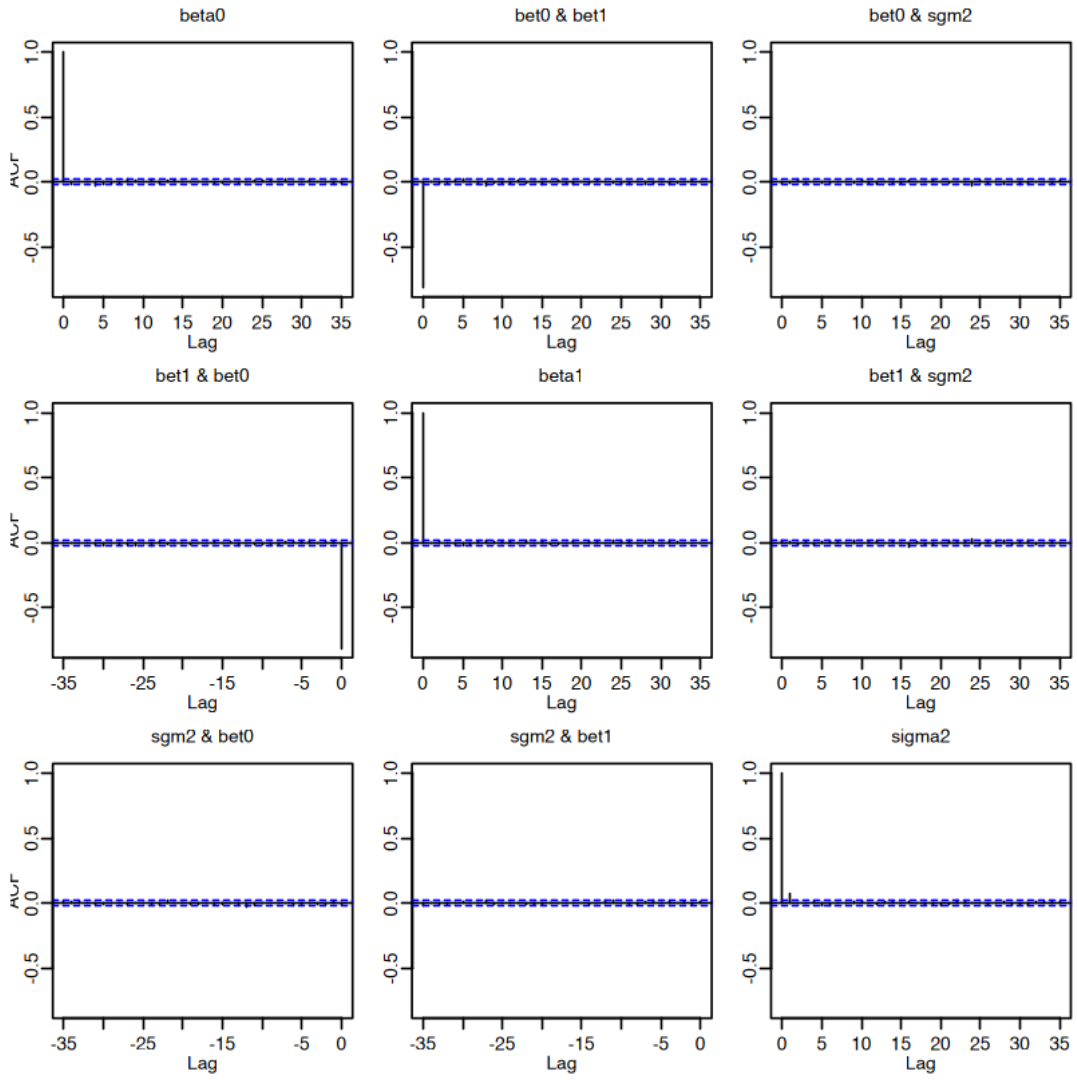


Oppure potremmo usare ggplot

[9]:
```
data_plot_mcmc_base <- data.frame(iterations = 1:
 ↪nrow(M1_post_samples$beta_samples), beta0 = M1_post_samples$beta_samples[,␣
 ↪1], beta1 = M1_post_samples$beta_samples[, 2], sigma2 =␣
 ↪M1_post_samples$sigma2_samples)

data_plot_mcmc <- data_plot_mcmc_base%>% pivot_longer(
  cols = -iterations,
    names_to = "parameters",
  values_to = "value"
)
data_plot_mcmc %>% ggplot(aes(x = iterations, y = value)) +
```

```
  geom_line() + facet_wrap(~parameters, scales = "free_y")

acf(data_plot_mcmc_base[-1])
```





Adesso calcoliamo laa psoteriori di μ, per un valore qualsiasi di x, che è la a posteriori di

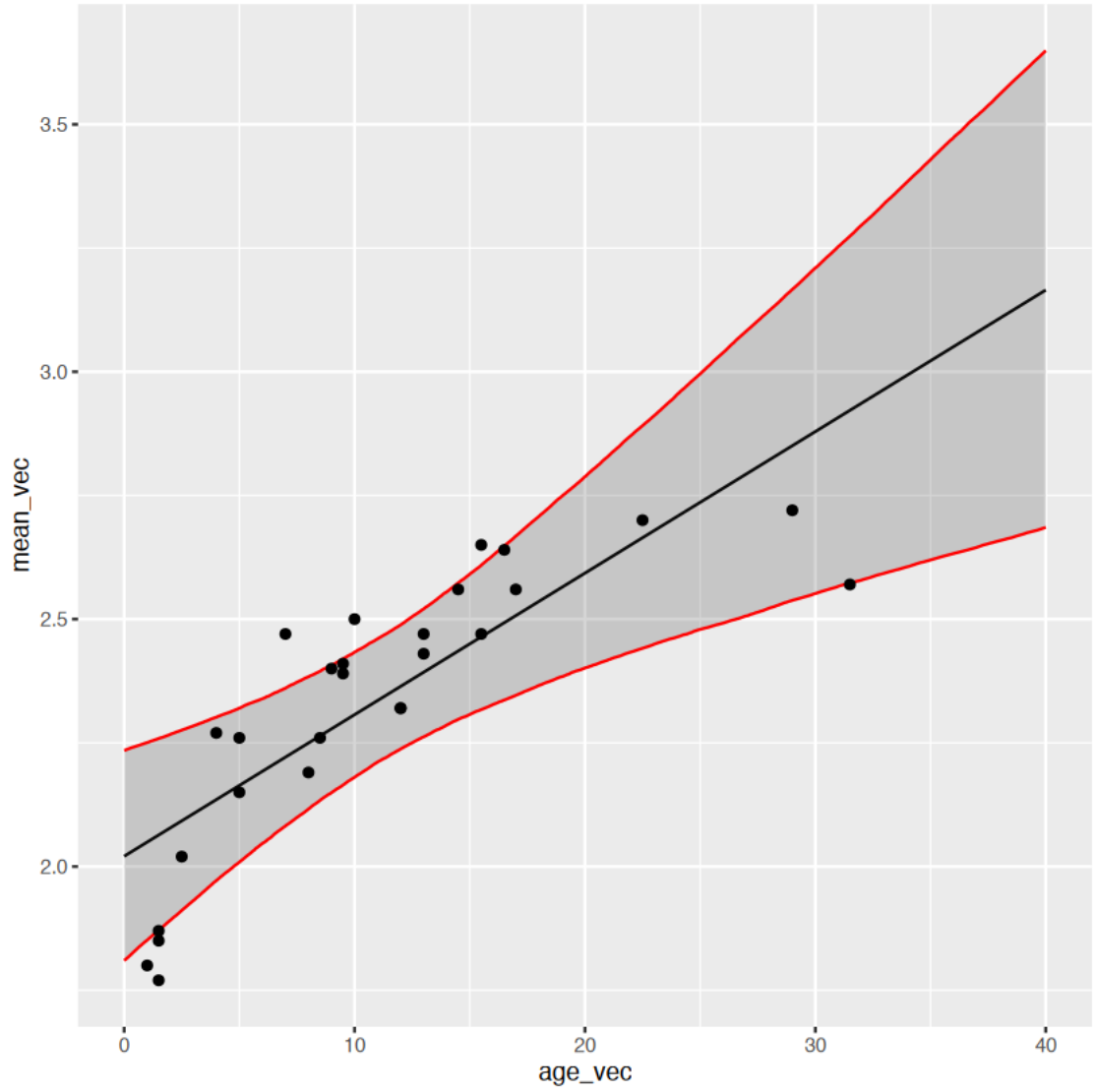
$$\beta_0 + \beta_i x$$

e ne calcoliamo la media e liinterallo di credibilità

[10]:
```
age_vec <- seq(0, 40 , by = 0.01)
mean_vec <- rep(0, length(age_vec))
q1_vec <- rep(0, length(age_vec))
q2_vec <- rep(0, length(age_vec))
for(ix in 1:length(age_vec))
{
  sim <- M1_post_samples$beta_samples[, 1] +␣
 ↪M1_post_samples$beta_samples[,2]*age_vec[ix]
```

```
  mean_vec[ix] <- mean(sim)
  q1_vec[ix] <- quantile(sim, probs = 0.025)
  q2_vec[ix] <- quantile(sim, probs = 1-0.025)

}
```

[11]:
```
data_plot <- data.frame(age_vec = age_vec, mean_vec = mean_vec, q1_vec =␣
 ↪q1_vec, q2_vec = q2_vec)

data_plot %>% ggplot(aes(x = age_vec, y = mean_vec)) +  geom_line() +
geom_ribbon(aes(ymin = q1_vec, ymax = q2_vec), alpha= 0.2, col="red") +
geom_point(data = data.frame(Y = Y, x = x), aes(y =Y, x = x))
```



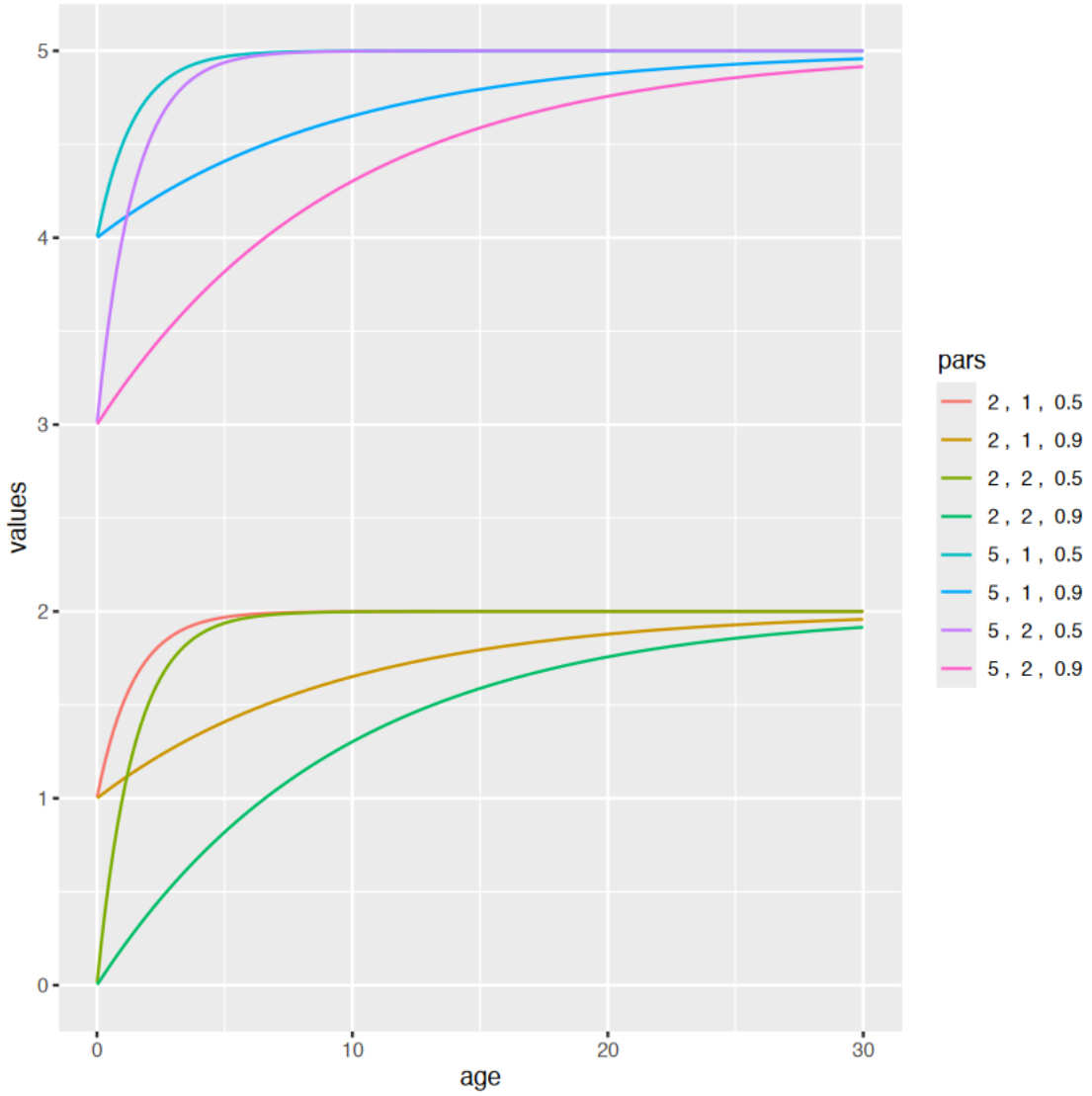
Non è sicuramente il modello migliore che potevamo fare. Proviama a migliorarlo

2. Assumete che la media sia $\beta_0 - \beta_1\gamma^{x_i}$ con $ [0,1], _0 > 0, _2 < _1 $ e implementato un modello per stimare la curva di crescita e la sua distribuzione, con approriate prior

per prima cosa vediamo perchè questo modello ha senso con dei plot

[12]:
```
# costruiamo il data frame iterativamente questa volta, con valori casuali
f_seq = seq(0.01, 30, by = 0.01) # punti dove valutiamo la funzione
data_function <- data.frame(age = 0, values = 0, pars = "0", beta0 = 0, beta1 =␣
↪0, gamma = 0)
for(beta0 in c(2,5))
{
  for(beta1 in c(1,2))
  {
    for(gamma in c(0.5,0.9))
    {
      dd <- data.frame(age = f_seq, values = beta0 -beta1*gamma^(f_seq), pars =␣
↪paste(beta0,", ", beta1,", ", gamma), beta0 = beta0, beta1 = beta1, gamma =␣
↪gamma)
      data_function <- rbind(data_function, dd)
    }
  }
}
data_function <- data_function[-1, ]
```

[13]:
```
data_function %>%  ggplot(aes(x = age, y = values, col= pars)) +geom_line()
```



In termine di prior potremmo fare mettere nelle prior i vari vincoli, oppure lasciare che il modello scelga quali valori sono appropriati. Io decido di avere $\beta_j \sim N(0, \tau^2)$ e $\sigma^2 \sim IG(a, b)$ e $\gamma \sim U(0.5, 1)$ (abbiamo pochi dati, e γ è difficilmente stimabile). Per γ utilizzo una proposal Uniforme

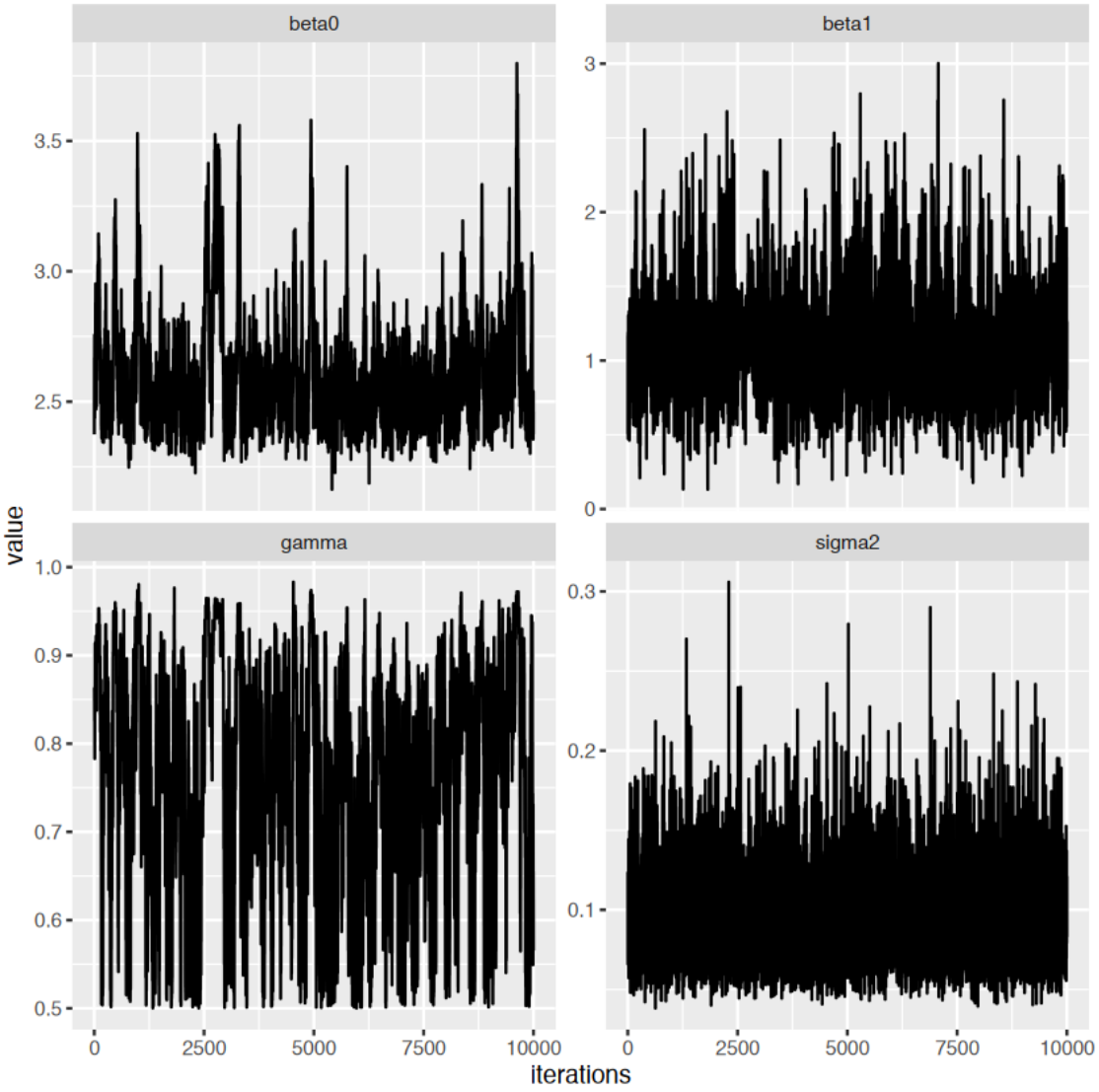
[14]:
```
bayesian_mcmc_with_gamma <- function(Y, X, tau2 = 10, a = 2, b = 1, n_iter =␣
↪10000, a_gamma = 0.5, b_gamma = 1) {
 # Number of data points
 n <- length(Y)
 # Initialize vectors for samples
 beta_0_samples <- numeric(n_iter)
 beta_1_samples <- numeric(n_iter)
 sigma2_samples <- numeric(n_iter)
 gamma_samples <- numeric(n_iter)
```

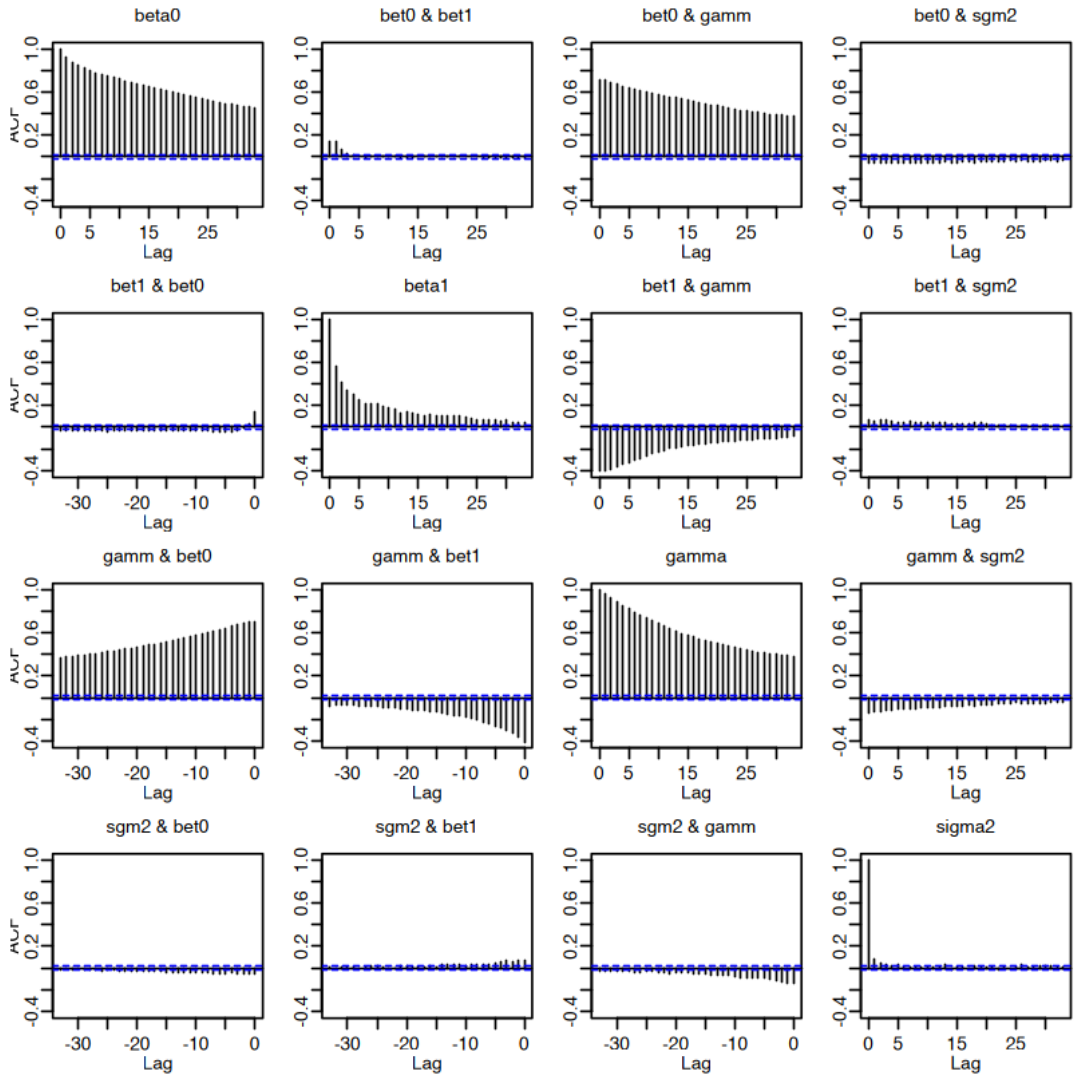
```r
# Initial values
beta_0 <- 3
beta_1 <- 1
sigma2 <- 1
gamma <- runif(1, a_gamma, b_gamma)

# MCMC sampling
for (i in 1:n_iter) {
  # Update beta_0 (conditional on beta_1, gamma, x, Y, sigma^2)
  V_beta_0 <- solve(n / sigma2 + 1 / tau2)
  m_beta_0 <- V_beta_0 %*% (sum(Y + beta_1 * gamma^X) / sigma2)
  beta_0 <- mvrnorm(1, m_beta_0, V_beta_0)

  # Update beta_1 (conditional on beta_0, gamma, x, Y, sigma^2)
  V_beta_1 <- solve(sum((gamma^X)^2) / sigma2 + 1 / tau2)
  m_beta_1 <- V_beta_1 %*% sum(gamma^X *  (-Y+beta_0)) / sigma2
  beta_1 <- mvrnorm(1, m_beta_1, V_beta_1)

  # Update sigma^2 (Inverse Gamma prior)
  residuals <- Y - (beta_0 - beta_1 * gamma^X)
  shape <- a + n / 2
  rate <- b + sum(residuals^2) / 2
  sigma2 <- rinvgamma(1, shape, rate)

  # Update gamma (using Metropolis-Hastings)
  gamma_proposal <- runif(1, max(gamma - 0.1, a_gamma), min(gamma + 0.1,␣
↪b_gamma)) # distribuzione Q
  residuals_proposal <- Y - (beta_0 - beta_1 * gamma_proposal^X)
  log_likelihood_proposal <- -0.5 * sum((residuals_proposal)^2) / sigma2
  residuals_current <- Y - (beta_0 - beta_1 * gamma^X)
  log_likelihood_current <- -0.5 * sum((residuals_current)^2) / sigma2

  log_q_proposal <- log(1 / (min(gamma + 0.1, b_gamma) - max(gamma - 0.1,␣
↪a_gamma)))
  log_q_current <- log(1 / (min(gamma_proposal + 0.1, b_gamma) -␣
↪max(gamma_proposal - 0.1, a_gamma)))

  log_ratio_numeratore <- log_likelihood_proposal + log_q_current
  log_ratio_denominatore <- log_likelihood_current + log_q_proposal

  if (runif(1,0,1) < exp(log_ratio_numeratore - log_ratio_denominatore)) {
    gamma <- gamma_proposal
  }

  # Save the samples
  beta_0_samples[i] <- beta_0
```

```
    beta_1_samples[i] <- beta_1
    sigma2_samples[i] <- sigma2
    gamma_samples[i] <- gamma
  }

  # Return the results
  list(
    beta_0_samples = beta_0_samples,
    beta_1_samples = beta_1_samples,
    sigma2_samples = sigma2_samples,
    gamma_samples = gamma_samples
  )
}
```

[15]:
```
## in questo caso X è il vettore dell'età
M1_v2_post_samples <- bayesian_mcmc_with_gamma(Y = Y, X = x, tau2 = 10000, a =␣
↪1, b = 1, n_iter = 10000, a_gamma = 0.5, b_gamma = 1)
```

[16]:
```
data_plot_mcmc_base <- data.frame(iterations = 1:
↪length(M1_v2_post_samples$beta_0_samples), beta0 =␣
↪M1_v2_post_samples$beta_0_samples, beta1 =␣
↪M1_v2_post_samples$beta_1_samples, gamma = M1_v2_post_samples$gamma_samples,␣
↪sigma2 = M1_v2_post_samples$sigma2_samples)

data_plot_mcmc <- data_plot_mcmc_base %>% pivot_longer(
  cols = -iterations,
  names_to = "parameters",
  values_to = "value"
)
data_plot_mcmc %>% ggplot(aes(x = iterations, y = value)) +
  geom_line() +
  facet_wrap(~parameters, scales = "free_y")

# vediamo anceh l'acf
acf(data_plot_mcmc_base[, -1])
```

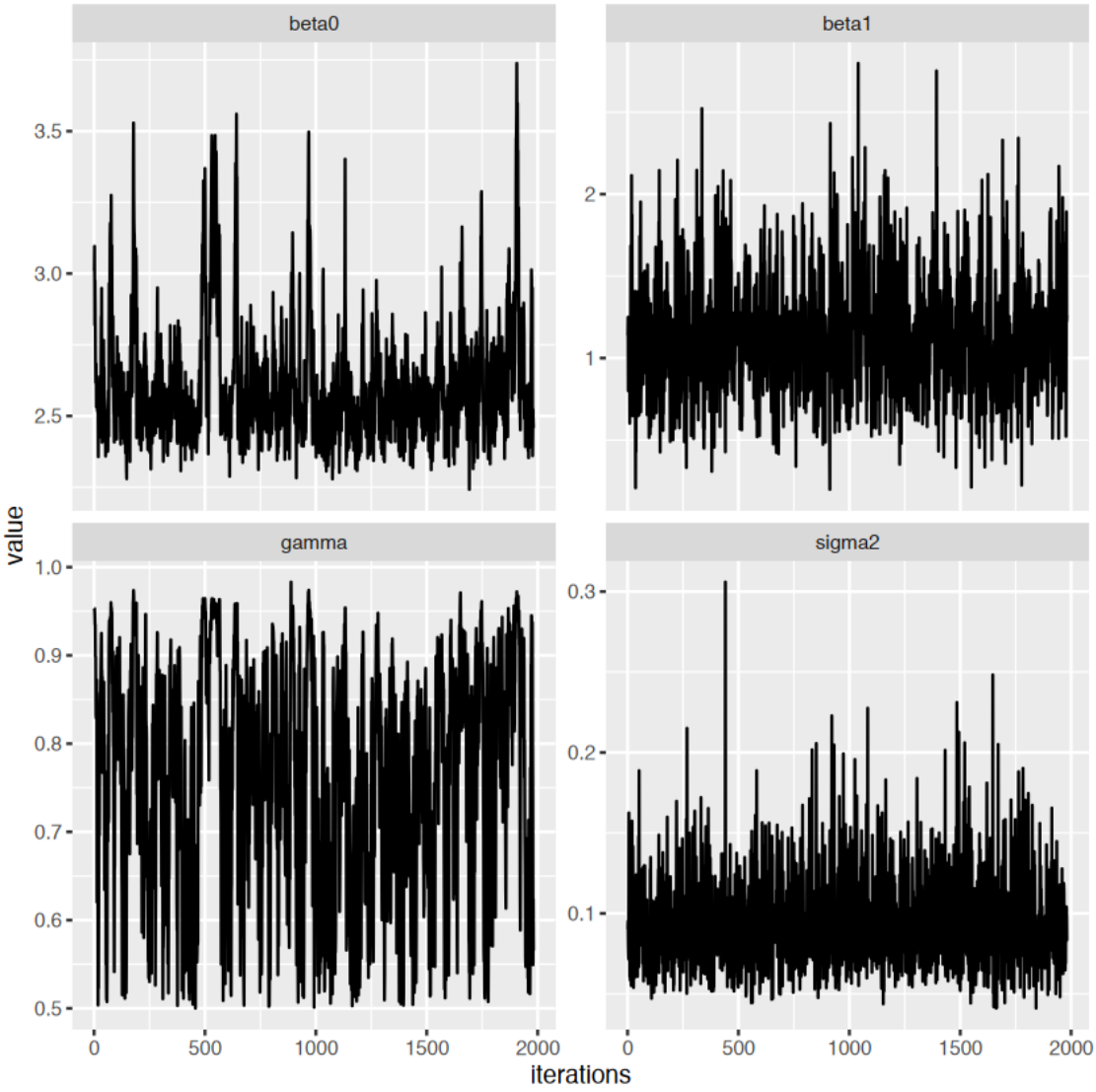

beta0
beta1
gamma
sigma2
value
iterations
3.5
3.0
2.5
3
2
1
0
1.0
0.9
0.8
0.7
0.6
0.5
0.3
0.2
0.1
0
2500
5000
7500
10000

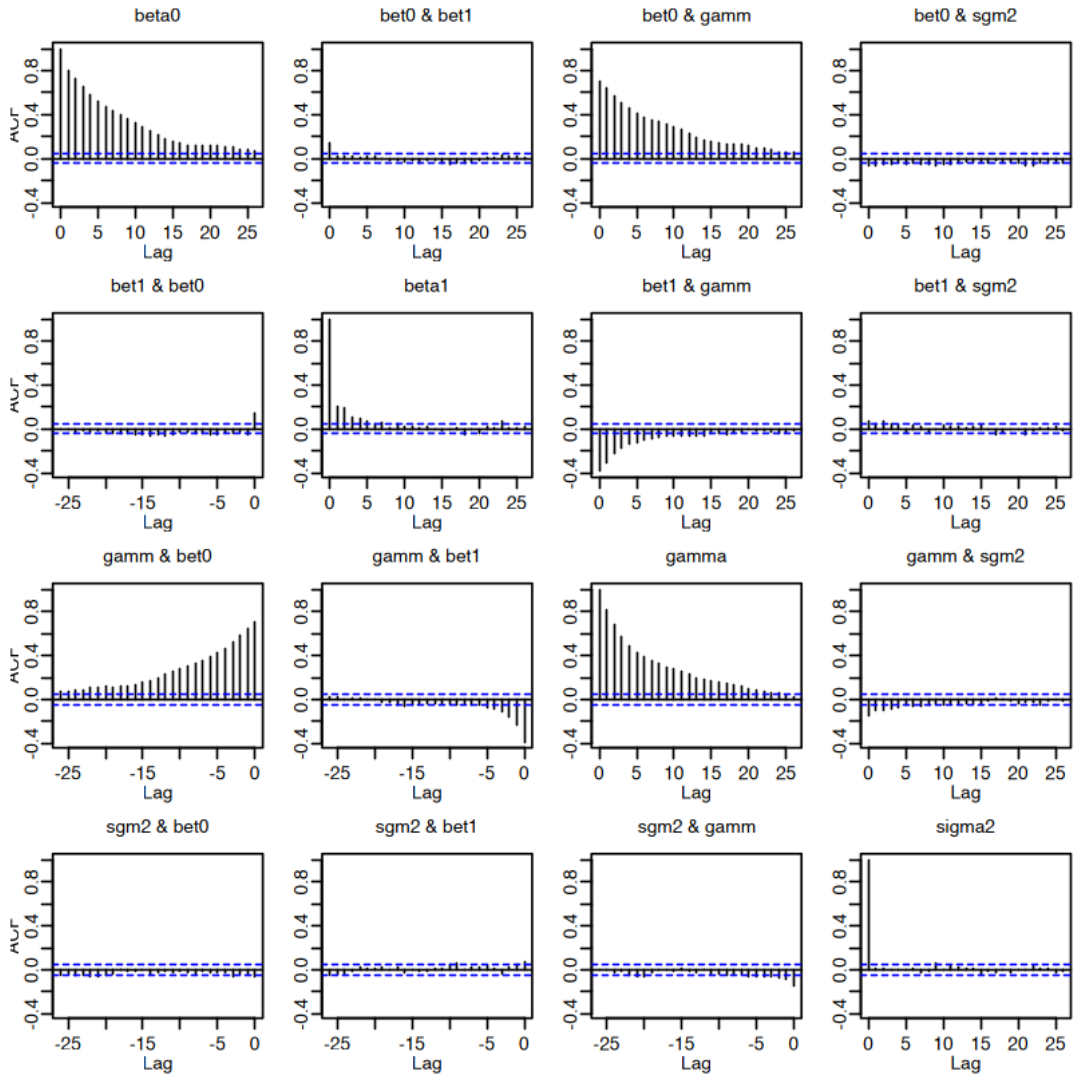


probabilmente ci vuole un po' di burnin e thin

[17]:
```
seq_mcmc_iter = seq(100, length(M1_v2_post_samples$beta_0_samples), by = 5)
M1_v2_post_samples$beta_0_samples <-␣
  ↪M1_v2_post_samples$beta_0_samples[seq_mcmc_iter]
M1_v2_post_samples$beta_1_samples <-␣
  ↪M1_v2_post_samples$beta_1_samples[seq_mcmc_iter]
M1_v2_post_samples$gamma_samples <-␣
  ↪M1_v2_post_samples$gamma_samples[seq_mcmc_iter]
M1_v2_post_samples$sigma2_samples <-␣
  ↪M1_v2_post_samples$sigma2_samples[seq_mcmc_iter]
```

[18]:
```r
data_plot_mcmc_base <- data.frame(iterations = 1:
  ↪length(M1_v2_post_samples$beta_0_samples), beta0 =␣
  ↪M1_v2_post_samples$beta_0_samples, beta1 =␣
  ↪M1_v2_post_samples$beta_1_samples, gamma = M1_v2_post_samples$gamma_samples,␣
  ↪sigma2 = M1_v2_post_samples$sigma2_samples)

data_plot_mcmc <- data_plot_mcmc_base %>% pivot_longer(
  cols = -iterations,
  names_to = "parameters",
  values_to = "value"
)
data_plot_mcmc %>% ggplot(aes(x = iterations, y = value)) +
  geom_line() +
  facet_wrap(~parameters, scales = "free_y")

# vediamo anceh l'acf
acf(data_plot_mcmc_base[, -1])
```


beta0
beta1
gamma
sigma2
value
iterations
3.5
3.0
2.5
2
1
1.0
0.9
0.8
0.7
0.6
0.5
0.3
0.2
0.1
0
500
1000
1500
2000

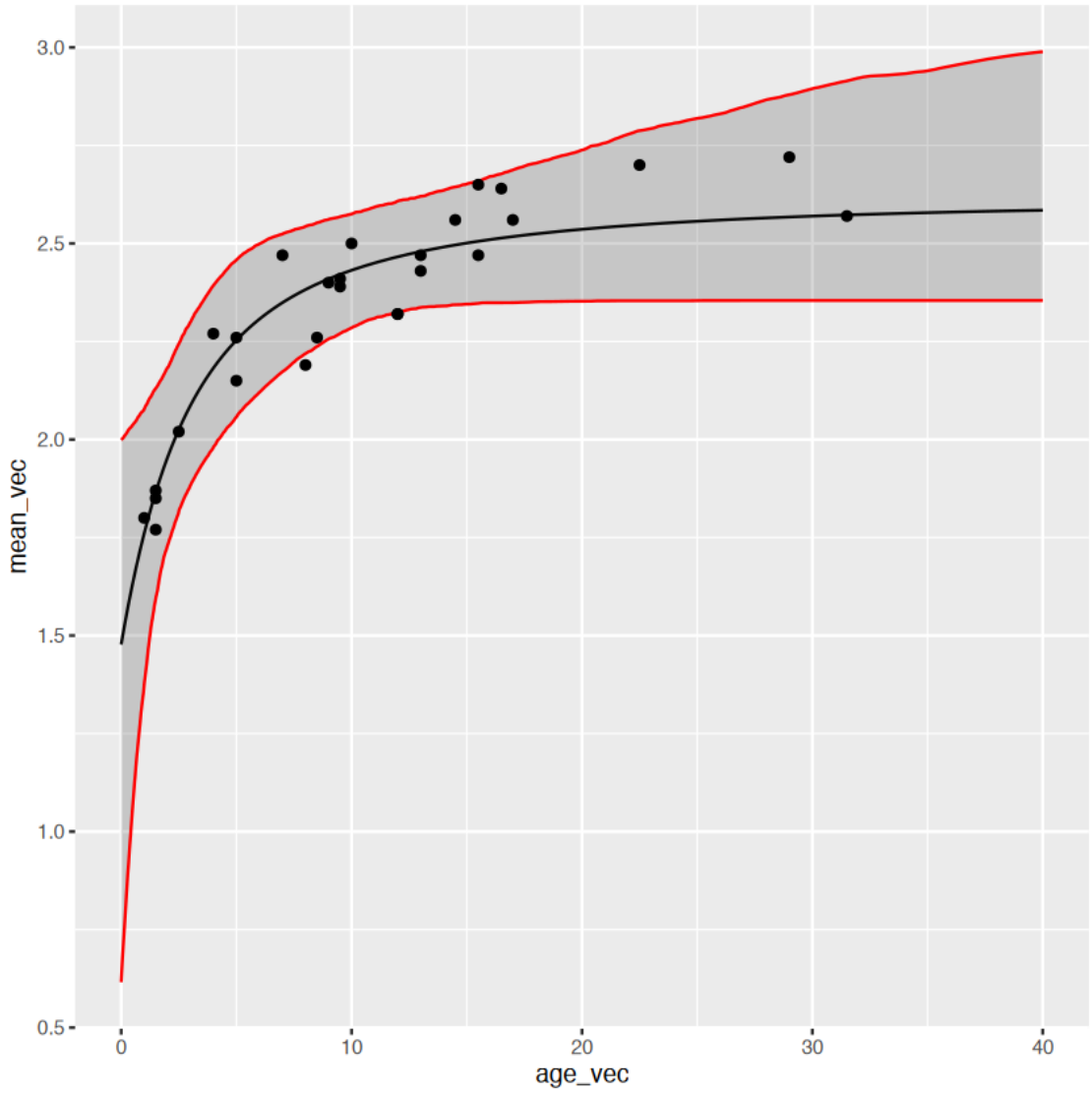


Le cose vanno un po' meglio in termine di diepndenza, ma non troppo. Adesso plottiamo come prima i valori della media

[19]:
```
age_vec <- seq(0, 40, by = 0.01)
mean_vec <- rep(0, length(age_vec))
q1_vec <- rep(0, length(age_vec))
q2_vec <- rep(0, length(age_vec))
for (ix in 1:length(age_vec))
{
  sim <- M1_v2_post_samples$beta_0_samples - M1_v2_post_samples$beta_1_samples␣
 ↪* M1_v2_post_samples$gamma^age_vec[ix]
  mean_vec[ix] <- mean(sim)
  q1_vec[ix] <- quantile(sim, probs = 0.025)
```

```
  q2_vec[ix] <- quantile(sim, probs = 1 - 0.025)
}
```

[20]:
```
data_plot <- data.frame(age_vec = age_vec, mean_vec = mean_vec, q1_vec =␣
↪q1_vec, q2_vec = q2_vec)

data_plot %>% ggplot(aes(x = age_vec, y = mean_vec)) +
  geom_line() +
  geom_ribbon(aes(ymin = q1_vec, ymax = q2_vec), alpha = 0.2, col = "red") +
  geom_point(data = data.frame(Y = Y, x = x), aes(y = Y, x = x))
```



3. Rifate il punto 2 assumendo M2 come verosimiglianza, e proponete tutti i parametri con una Normale (univariata o Multivariata), adattando la varianza. Confrontate i risultati, le curve

di crescita, e utilizzando qualche metrica (CRPS, MSE, AIC) decidete quale dei due è migliore

Decico di proporre da normali univariate. Per i parametri β non c'è problema, mentre, per avere tutti i parametri su $\mathbb{R}$, propongo

$$\rho = log(\sigma^2) \Rightarrow \sigma^2 = \exp(\rho)$$

e

$$\psi = log(\frac{\gamma - a_\gamma}{b_\gamma - \gamma}) \Rightarrow \frac{a_\gamma + b_\gamma exp(\psi)}{1 + exp(\psi)}$$

dove $\gamma \sim U(a_\gamma, b_\gamma)$. Ho che

$$f_\rho(\rho) = f_{\sigma^2}(\sigma^2(\rho))|\frac{d\sigma^2(\rho)}{d\rho}| = f_{\sigma^2}(\sigma^2(\rho))\exp(\rho) = f_{\sigma^2}(\sigma^2(\rho))\sigma^2(\rho)$$

e

$$f_\psi(\psi) = f_\gamma(\gamma(\psi))|\frac{d\gamma(\psi)}{d\psi}| = f_\gamma(\gamma(\psi))\frac{(b-1)e^{(\psi)}}{(1+exp(\psi))^2}$$

In questo caso dobbiamo anche tenere in conto che se i parametri della Gamma sono negativi dobbiamo rifiutare perchè non ammessibile. Si può dimostrare che l'approxixo è valodi, ma se volete pensatela come se parametri negativi fossero possibili, ma danno luogo a una densità Infinitamente piccola, e quindi rifiutate sempre

[21]:
```r
# Function for MCMC sampling with Gamma likelihood
bayesian_mcmc_with_gamma_likelihood <- function(Y, X, tau2 = 10, a = 2, b = 1,␣
 ↪n_iter = 10000, a_gamma = 0.5, b_gamma = 1, nbatch = 50, A = 100, B = 1000,␣
 ↪sd_prop_beta_0 = 0.001, sd_prop_beta_1 = 0.001, sd_prop_gamma = 0.001,␣
 ↪sd_prop_sigma2 = 0.001, alpha_target = 0.234) {
  # Number of data points
  n <- length(Y)
  # Initialize vectors for samples
  beta_0_samples <- numeric(n_iter)
  beta_1_samples <- numeric(n_iter)
  sigma2_samples <- numeric(n_iter)
  gamma_samples <- numeric(n_iter)

  # Initial values
  beta_0 <- 3
  beta_1 <- 1.5
  sigma2 <- 1
  gamma <- runif(1, a_gamma, b_gamma)

  # qui salvo le somme dei  valori del rapporto metropolis
  alpha_beta_0 <- 0
  alpha_beta_1 <- 0
  alpha_sigma2 <- 0
  alpha_gamma <- 0

  # MCMC sampling
```

```
count_iter <- 0
for (i in 1:n_iter) {
  count_iter <- count_iter +1

  # Update beta_0
  beta_0_proposal <- rnorm(1, beta_0, sd_prop_beta_0)
  mu_proposal <- beta_0_proposal - beta_1*gamma^X
  sigma2_proposal <- sigma2
  mu_current <- beta_0 - beta_1 * gamma^X
  sigma2_current <-  sigma2

  par_a_proposal <- mu_proposal / sigma2_proposal
  par_b_proposal <- 1 / sigma2_proposal
  par_a_current <- mu_current / sigma2_current
  par_b_current <- 1 / sigma2_current

  if(sum(par_a_proposal<=0)==0)
  {
    log_prior_proposal <- dnorm(beta_0_proposal, 0, tau2^0.5, log = T)
    log_prior_current <- dnorm(beta_0, 0, tau2^0.5, log = T)
    ## la proposta è simmetrica e si annulla
    log_ratio_numeratore <- sum(dgamma(Y, shape = par_a_proposal, rate =␣
↪par_b_proposal, log = T)) + log_prior_proposal
    log_ratio_denominatore <- sum(dgamma(Y, shape = par_a_current, rate =␣
↪par_b_current, log = T)) + log_prior_current


    alpha_beta_0 <- alpha_beta_0 + min(exp(log_ratio_numeratore -␣
↪log_ratio_denominatore),1)
    if (runif(1, 0, 1) < exp(log_ratio_numeratore - log_ratio_denominatore)) {
      beta_0 <- beta_0_proposal
    }
  }else{
    alpha_beta_0 <- alpha_beta_0 + 0
  }

  # Update beta_1
  beta_1_proposal <- rnorm(1, beta_1, sd_prop_beta_1)
  mu_proposal <- beta_0 - beta_1_proposal * gamma^X
  sigma2_proposal <-  sigma2
  mu_current <- beta_0 - beta_1 * gamma^X
  sigma2_current <-   sigma2

  par_a_proposal <- mu_proposal / sigma2_proposal
  par_b_proposal <- 1 / sigma2_proposal
  par_a_current <- mu_current / sigma2_current
  par_b_current <- 1 / sigma2_current
```

```
  if(sum(par_a_proposal<=0)==0)
  {
    log_prior_proposal <- dnorm(beta_1_proposal, 0, tau2^0.5, log = T)
    log_prior_current <- dnorm(beta_1, 0, tau2^0.5, log = T)
    ## la proposta è simmetrica e si annulla
    log_ratio_numeratore <- sum(dgamma(Y, shape = par_a_proposal, rate =␣
↪par_b_proposal, log = T)) + log_prior_proposal
    log_ratio_denominatore <- sum(dgamma(Y, shape = par_a_current, rate =␣
↪par_b_current, log = T)) + log_prior_current

    alpha_beta_1 <- alpha_beta_1 + min(exp(log_ratio_numeratore -␣
↪log_ratio_denominatore), 1)
    if (runif(1, 0, 1) < exp(log_ratio_numeratore - log_ratio_denominatore)) {
      beta_1 <- beta_1_proposal
    }
  }else{
    alpha_beta_1 <- alpha_beta_1 + 0
  }


  # Update sigma2
  sigma2_proposal <- exp(rnorm(1, log(sigma2), sd_prop_sigma2))
  mu_proposal <- beta_0 - beta_1 * gamma^X
  sigma2_proposal <- sigma2_proposal
  mu_current <- beta_0 - beta_1 * gamma^X
  sigma2_current <- sigma2

  par_a_proposal <- mu_proposal / sigma2_proposal
  par_b_proposal <- 1 / sigma2_proposal
  par_a_current <- mu_current / sigma2_current
  par_b_current <- 1 / sigma2_current


  log_prior_proposal <- -(a + 1) * log(sigma2_proposal) - b / sigma2_proposal␣
↪+ log(sigma2_proposal)
  log_prior_current <- -(a + 1) * log(sigma2) - b / sigma2 + log(sigma2)
  ## la proposta è simmetrica e si annulla
  log_ratio_numeratore <- sum(dgamma(Y, shape = par_a_proposal, rate =␣
↪par_b_proposal, log = T)) + log_prior_proposal
  log_ratio_denominatore <- sum(dgamma(Y, shape = par_a_current, rate =␣
↪par_b_current, log = T)) + log_prior_current

  alpha_sigma2 <- alpha_sigma2 + min(exp(log_ratio_numeratore -␣
↪log_ratio_denominatore), 1)
```

```
  if (runif(1, 0, 1) < exp(log_ratio_numeratore - log_ratio_denominatore)) {
    sigma2 <- sigma2_proposal
  }

  # Update gamma
  psi <- log((gamma - a_gamma)/(b_gamma-gamma))

  psi_proposal <- rnorm(1, psi, sd_prop_gamma)
  gamma_proposal <- (a_gamma + b_gamma*exp(psi_proposal)) / (1 +␣
↪exp(psi_proposal))

  mu_proposal <- beta_0 - beta_1 * gamma_proposal^X
  sigma2_proposal <-   sigma2
  mu_current <- beta_0 - beta_1 * gamma^X
  sigma2_current <-  sigma2


  par_a_proposal <- mu_proposal / sigma2_proposal
  par_b_proposal <- 1 / sigma2_proposal
  par_a_current <- mu_current / sigma2_current
  par_b_current <- 1 / sigma2_current


  if(sum(par_a_proposal<=0)==0)
  {
    log_prior_proposal <- psi_proposal - 2 * log(1 + exp(psi_proposal))
    log_prior_current <- psi - 2 * log(1 + exp(psi))
    ## la proposta è simmetrica e si annulla
    log_ratio_numeratore <- sum(dgamma(Y, shape = par_a_proposal, rate =␣
↪par_b_proposal, log = T)) + log_prior_proposal

    log_ratio_denominatore <- sum(dgamma(Y, shape = par_a_current, rate =␣
↪par_b_current, log = T)) + log_prior_current


    alpha_gamma <- alpha_gamma + min(exp(log_ratio_numeratore -␣
↪log_ratio_denominatore), 1)
    if (runif(1, 0, 1) < exp(log_ratio_numeratore - log_ratio_denominatore)) {
      gamma <- gamma_proposal
    }


  }else{
      alpha_gamma <- alpha_gamma + 0
  }
  ## adapt variance (sarebbe meglio non farla dopo il burnin)
  if (count_iter %% nbatch == 0) {
```

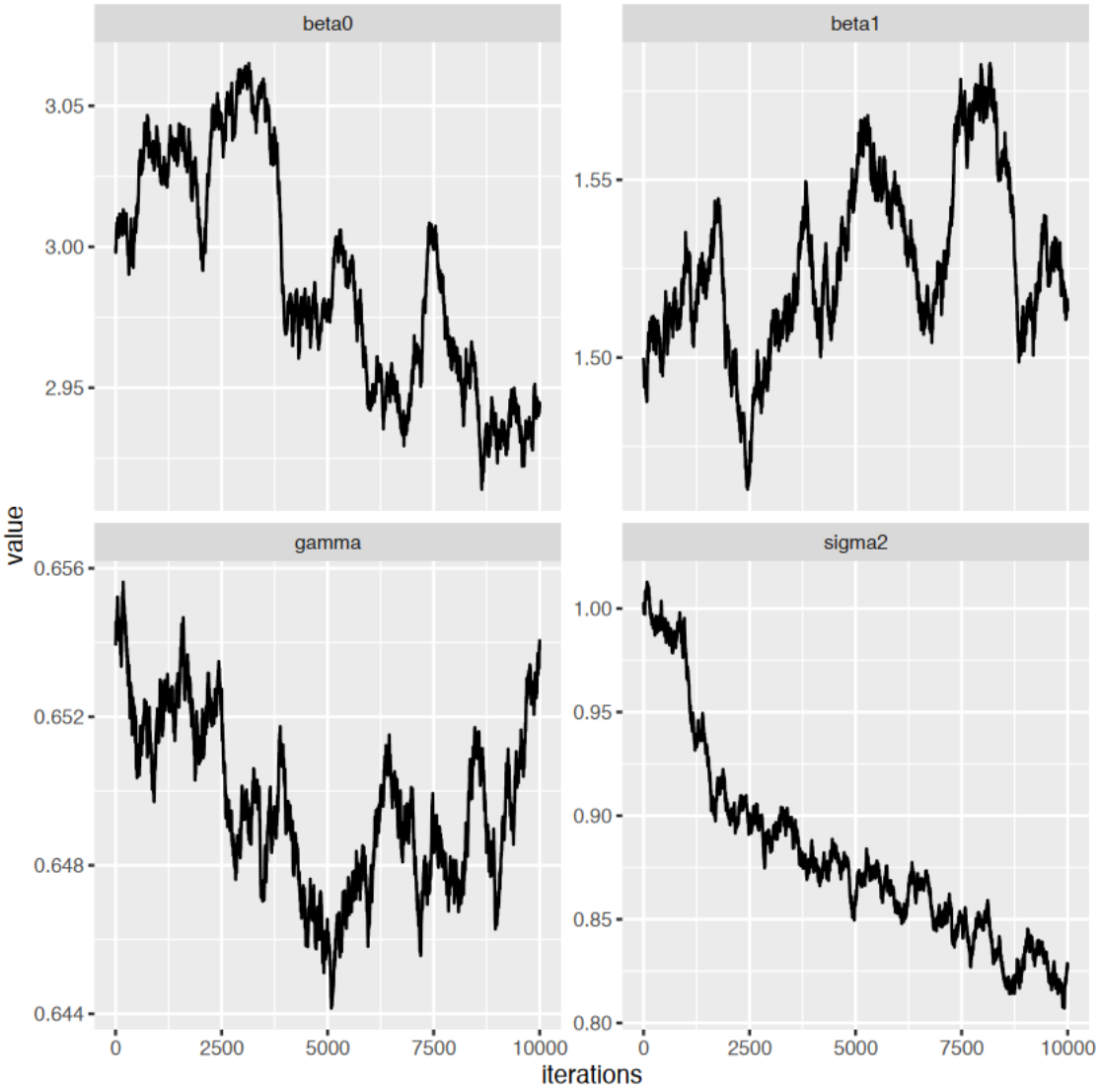
```
      alpha_beta_0 <- alpha_beta_0 / nbatch
      alpha_beta_1 <- alpha_beta_1 / nbatch
      alpha_sigma2 <- alpha_sigma2 / nbatch
      alpha_gamma <- alpha_gamma / nbatch

      sd_prop_beta_0 <- exp(log(sd_prop_beta_0) + A / (B + count_iter) *␣
 ↪(alpha_beta_0 - alpha_target))
      sd_prop_beta_1 <- exp(log(sd_prop_beta_1) + A / (B + count_iter) *␣
 ↪(alpha_beta_1 - alpha_target))
      sd_prop_sigma2 <- exp(log(sd_prop_sigma2) + A / (B + count_iter) *␣
 ↪(alpha_sigma2 - alpha_target))
      sd_prop_gamma <- exp(log(sd_prop_gamma) + A / (B + count_iter) *␣
 ↪(alpha_gamma - alpha_target))

      alpha_beta_0 <- 0
      alpha_beta_1 <- 0
      alpha_sigma2 <- 0
      alpha_gamma <- 0
    }


    # Save the samples
    beta_0_samples[i] <- beta_0
    beta_1_samples[i] <- beta_1
    sigma2_samples[i] <- sigma2
    gamma_samples[i] <- gamma
  }

  # Return the results
  list(
    beta_0_samples = beta_0_samples,
    beta_1_samples = beta_1_samples,
    sigma2_samples = sigma2_samples,
    gamma_samples = gamma_samples
  )
}
```

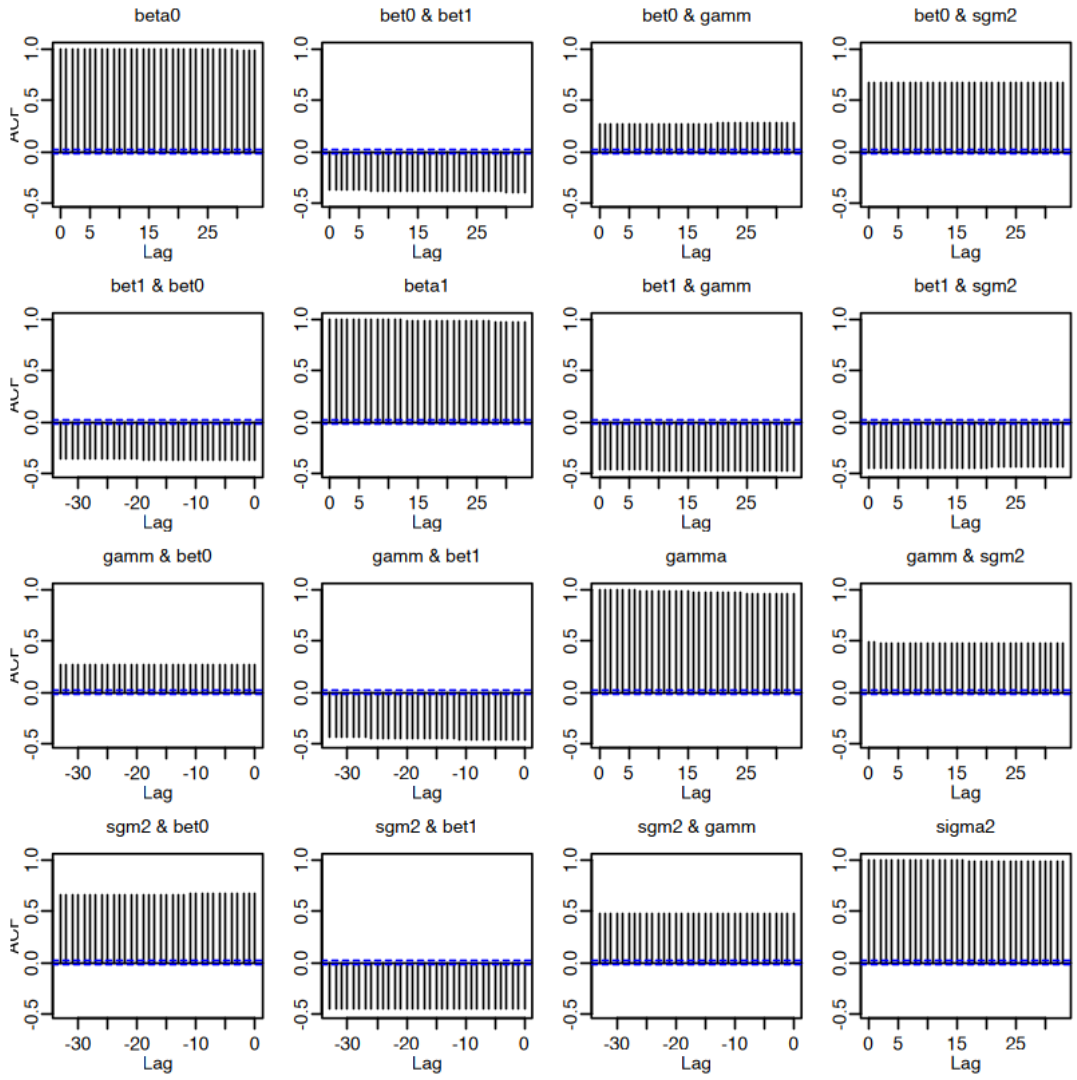
Proviamo l'algoritmo senza fare adapt (nbatch più alto del numero di iterazioni)

[22]:
```
set.seed(100)
M2_post_samples <- bayesian_mcmc_with_gamma_likelihood(Y = Y, X = x, tau2 =␣
 ↪10000, a = 1, b = 1, n_iter = 10000, a_gamma = 0.5, b_gamma = 1, nbatch =␣
 ↪500000000, A = 100, B = 1000, sd_prop_beta_0 = 0.001, sd_prop_beta_1 = 0.
 ↪001, sd_prop_gamma = 0.001, sd_prop_sigma2 = 0.001, alpha_target = 0.234)
```

[23]:
```
```

```r
data_plot_mcmc_base <- data.frame(iterations = 1:
  ↪length(M2_post_samples$beta_0_samples), beta0 =␣
  ↪M2_post_samples$beta_0_samples, beta1 = M2_post_samples$beta_1_samples,␣
  ↪gamma = M2_post_samples$gamma_samples, sigma2 =␣
  ↪M2_post_samples$sigma2_samples)

data_plot_mcmc <- data_plot_mcmc_base %>% pivot_longer(
  cols = -iterations,
  names_to = "parameters",
  values_to = "value"
)
data_plot_mcmc %>% ggplot(aes(x = iterations, y = value)) +
  geom_line() +
  facet_wrap(~parameters, scales = "free_y")

# vediamo anceh l'acf
acf(data_plot_mcmc_base[, -1])
```


beta0
beta1
gamma
sigma2
value
iterations
3.05
3.00
2.95
1.55
1.50
0.656
0.652
0.648
0.644
1.00
0.95
0.90
0.85
0.80
0
2500
5000
7500
10000

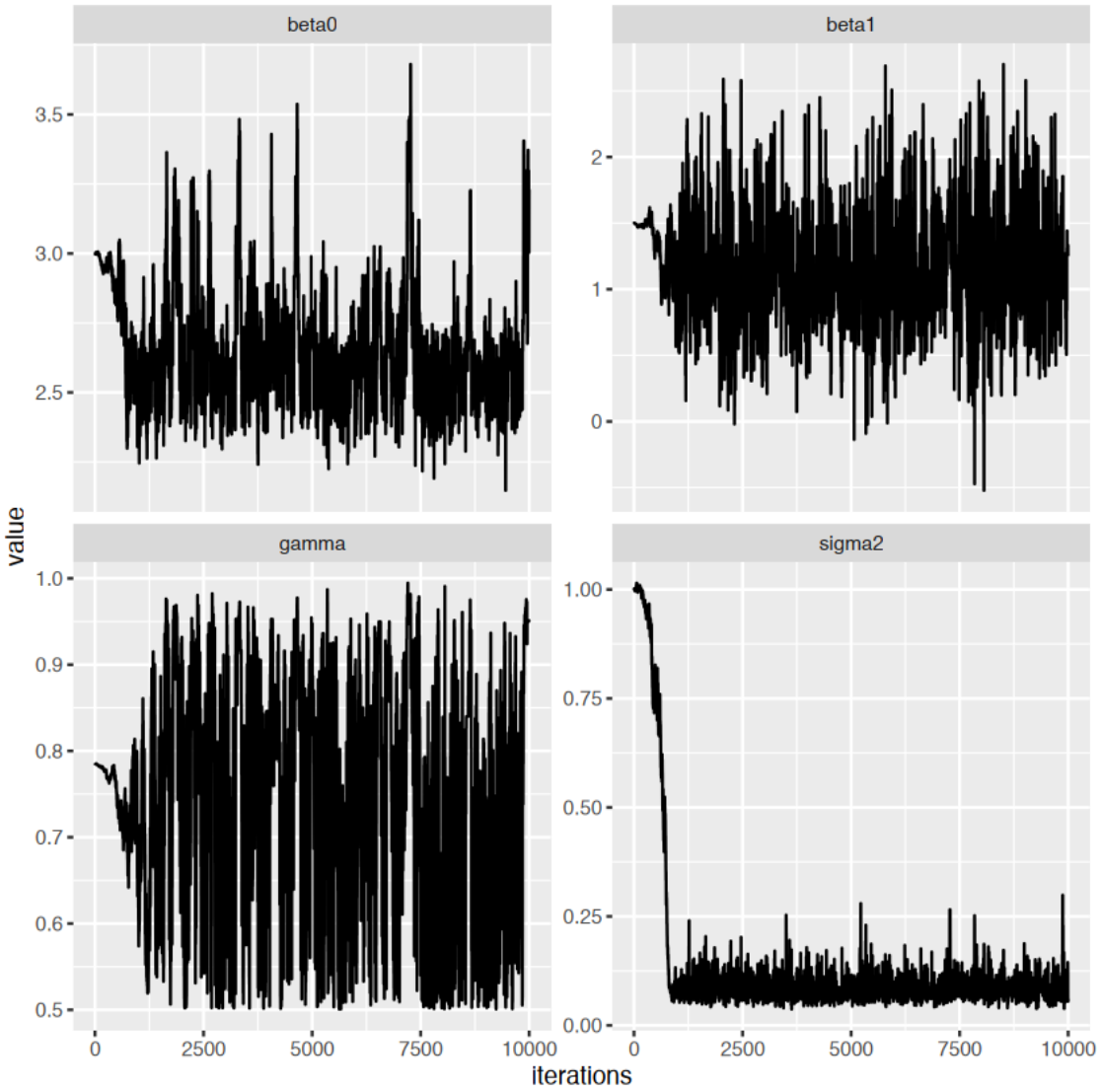


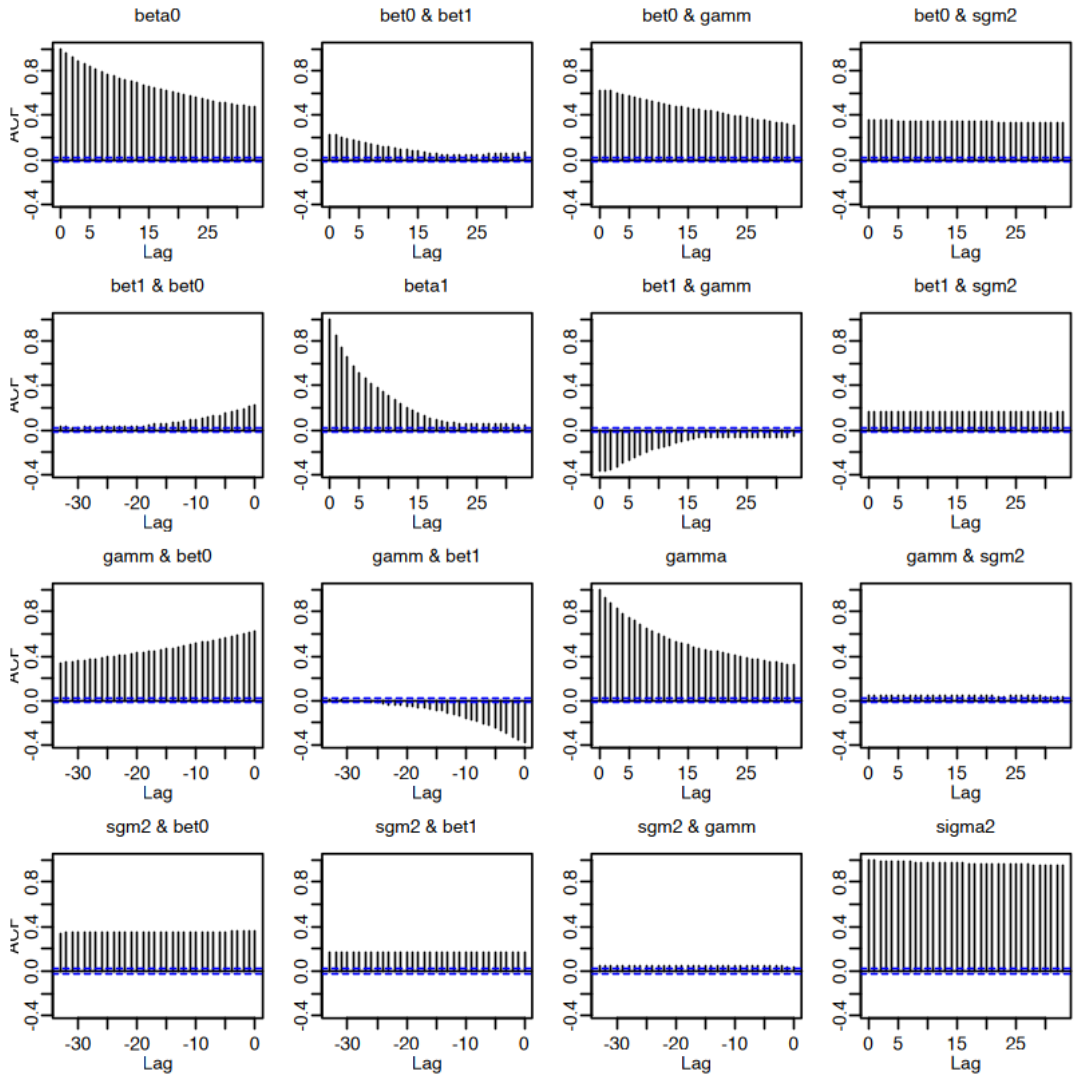
mettiamo adesso un nbatch di 50

[24]:
```
M2_post_samples <- bayesian_mcmc_with_gamma_likelihood(Y = Y, X = x, tau2 =␣
↪10000, a = 1, b = 1, n_iter = 10000, a_gamma = 0.5, b_gamma = 1, nbatch =␣
↪50, A = 500, B = 1000, sd_prop_beta_0 = 0.001, sd_prop_beta_1 = 0.001,␣
↪sd_prop_gamma = 0.001, sd_prop_sigma2 = 0.001, alpha_target = 0.234)
```

[25]:
```
data_plot_mcmc_base <- data.frame(iterations = 1:
↪length(M2_post_samples$beta_0_samples), beta0 =␣
↪M2_post_samples$beta_0_samples, beta1 = M2_post_samples$beta_1_samples,␣
↪gamma = M2_post_samples$gamma_samples, sigma2 =␣
↪M2_post_samples$sigma2_samples)
```

```r
data_plot_mcmc <- data_plot_mcmc_base %>% pivot_longer(
  cols = -iterations,
  names_to = "parameters",
  values_to = "value"
)
data_plot_mcmc %>% ggplot(aes(x = iterations, y = value)) +
  geom_line() +
  facet_wrap(~parameters, scales = "free_y")

# vediamo anceh l'acf
acf(data_plot_mcmc_base[, -1])
```



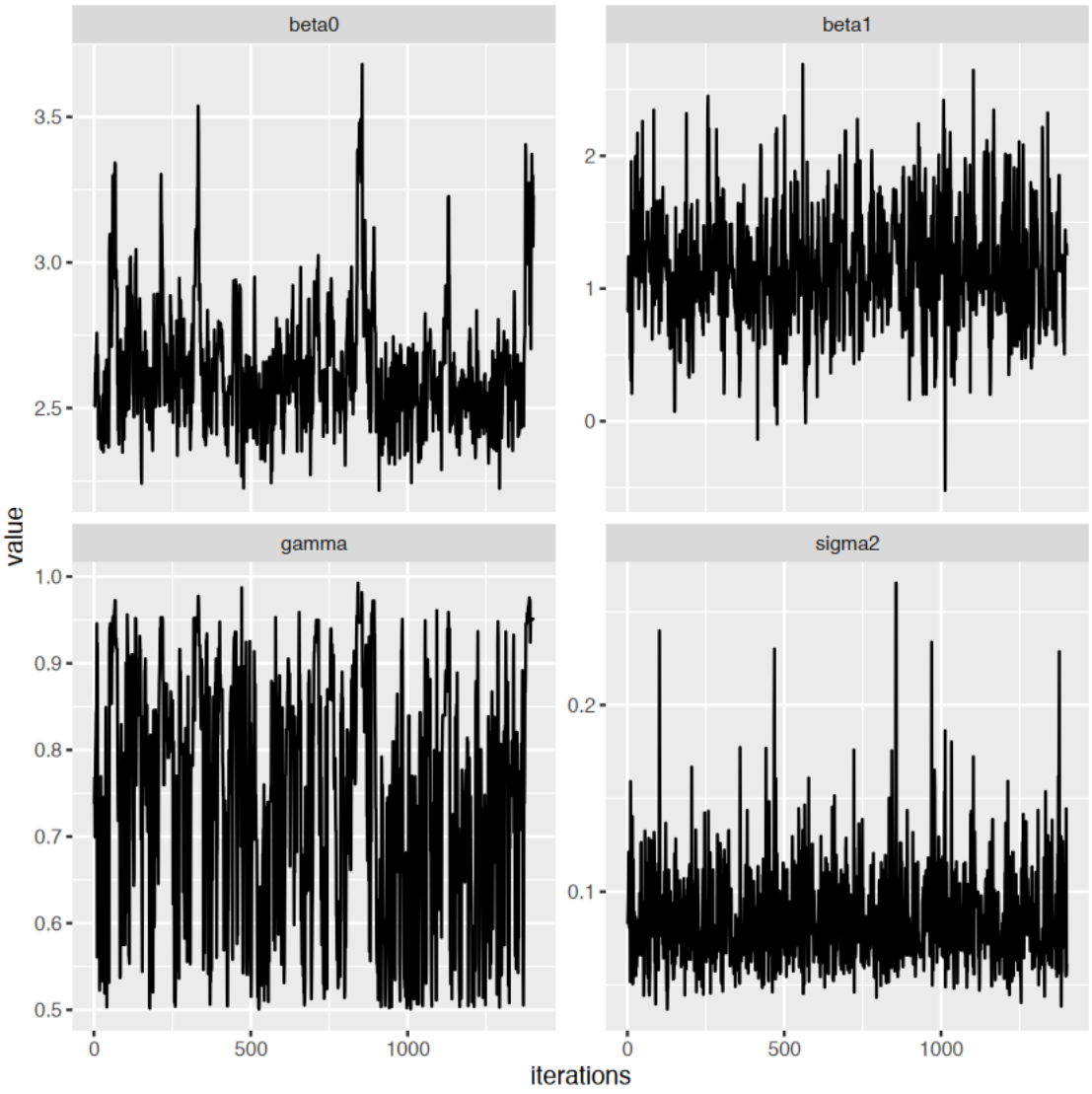


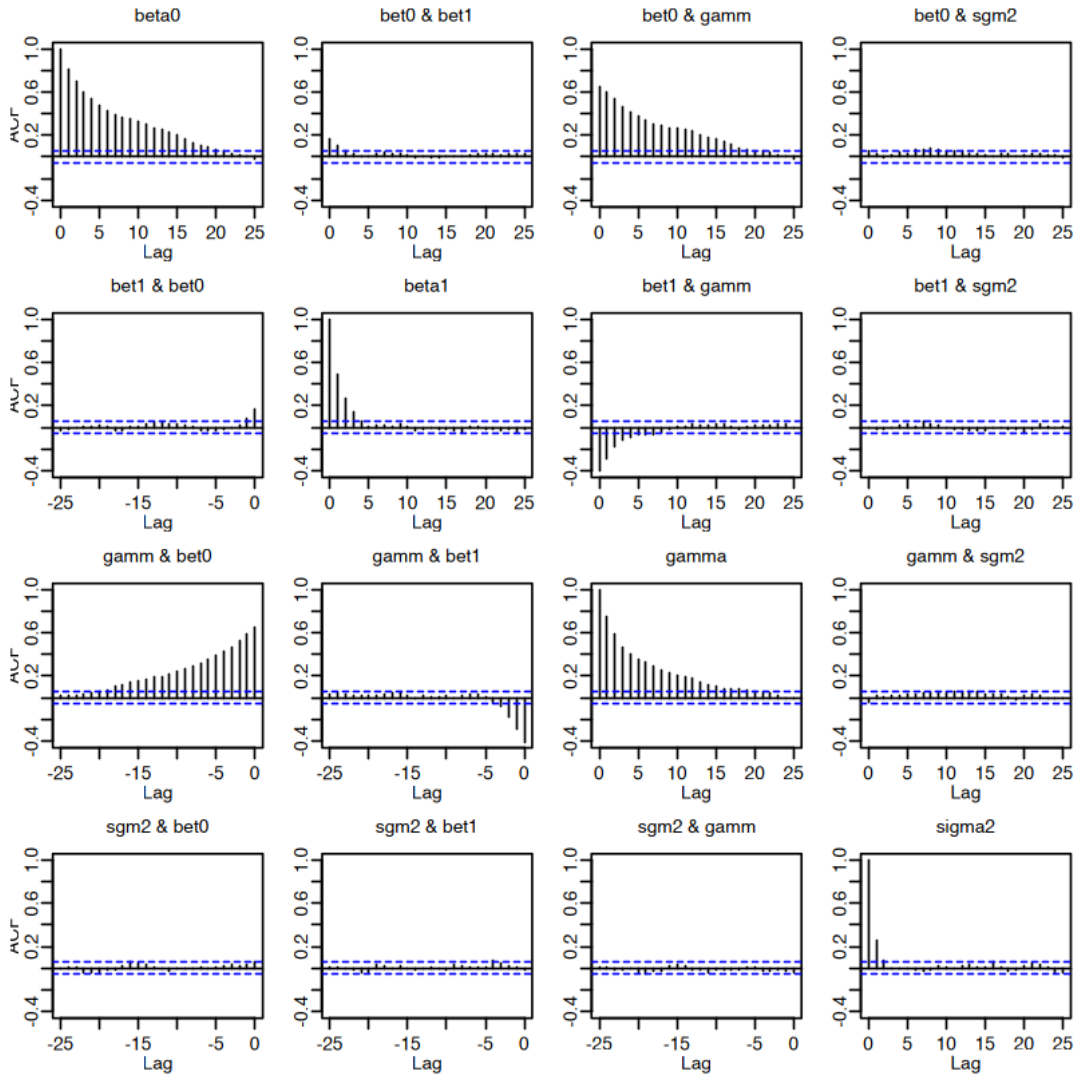
Le cose vanno meglio anceh se bisogna togliere il burnin e forse fare del thin

[26]:
```
seq_mcmc_iter <- seq(3000, length(M2_post_samples$beta_0_samples), by = 5)
M2_post_samples$beta_0_samples <- M2_post_samples$beta_0_samples[seq_mcmc_iter]
M2_post_samples$beta_1_samples <- M2_post_samples$beta_1_samples[seq_mcmc_iter]
M2_post_samples$gamma_samples <- M2_post_samples$gamma_samples[seq_mcmc_iter]
M2_post_samples$sigma2_samples <- M2_post_samples$sigma2_samples[seq_mcmc_iter]
```

[27]:
```
data_plot_mcmc_base <- data.frame(iterations = 1:
  ↪length(M2_post_samples$beta_0_samples), beta0 =␣
  ↪M2_post_samples$beta_0_samples, beta1 = M2_post_samples$beta_1_samples,␣
  ↪gamma = M2_post_samples$gamma_samples, sigma2 =␣
  ↪M2_post_samples$sigma2_samples)
```

```r
data_plot_mcmc <- data_plot_mcmc_base %>% pivot_longer(
  cols = -iterations,
  names_to = "parameters",
  values_to = "value"
)
data_plot_mcmc %>% ggplot(aes(x = iterations, y = value)) +
  geom_line() +
  facet_wrap(~parameters, scales = "free_y")

# vediamo anceh l'acf
acf(data_plot_mcmc_base[, -1])
```



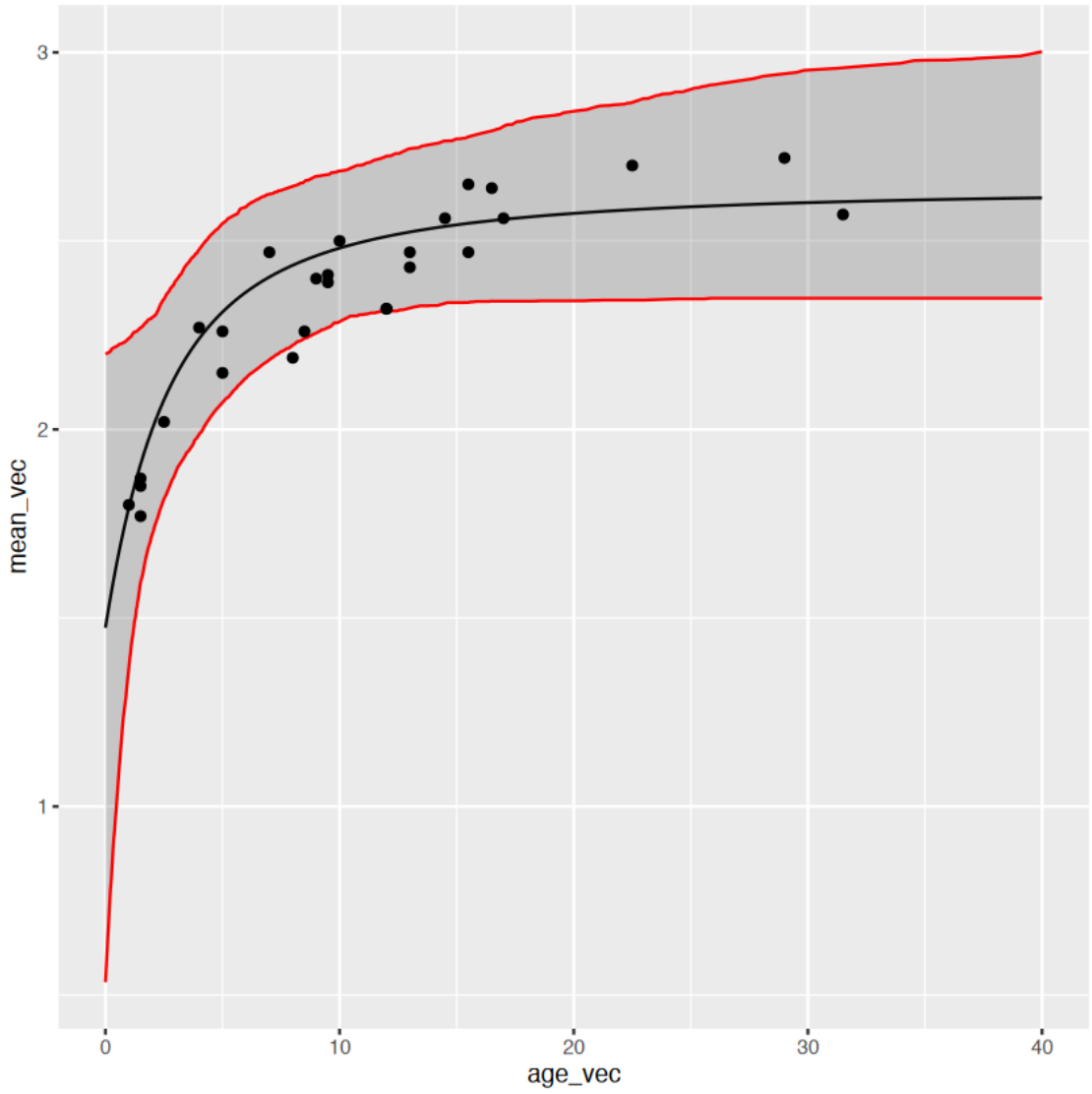


Calcoliamo adesso le curve di crescita

[28]:
```
age_vec <- seq(0, 40, by = 0.01)
mean_vec <- rep(0, length(age_vec))
q1_vec <- rep(0, length(age_vec))
q2_vec <- rep(0, length(age_vec))
for (ix in 1:length(age_vec))
{
  sim <- M2_post_samples$beta_0_samples - M2_post_samples$beta_1_samples *␣
 ↪M2_post_samples$gamma^age_vec[ix]
  mean_vec[ix] <- mean(sim)
  q1_vec[ix] <- quantile(sim, probs = 0.025)
  q2_vec[ix] <- quantile(sim, probs = 1 - 0.025)
```

```
}
```

[29]:
```
data_plot <- data.frame(age_vec = age_vec, mean_vec = mean_vec, q1_vec =␣
 ↪q1_vec, q2_vec = q2_vec)

data_plot %>% ggplot(aes(x = age_vec, y = mean_vec)) +
  geom_line() +
  geom_ribbon(aes(ymin = q1_vec, ymax = q2_vec), alpha = 0.2, col = "red") +
  geom_point(data = data.frame(Y = Y, x = x), aes(y = Y, x = x))
```



Vediamo adesso quale modello è preferibie. Calcoliamo l'AIC con il valor medio dei parametri

[30]:
```r
# M1
beta0_mean_M1 <- mean(M1_v2_post_samples$beta_0_samples)
beta1_mean_M1 <- mean(M1_v2_post_samples$beta_1_samples)
gamma_mean_M1 <- mean(M1_v2_post_samples$gamma_samples)
sigma2_mean_M1 <- mean(M1_v2_post_samples$sigma2_samples)

mean_normal_M1 <- beta0_mean_M1 - beta1_mean_M1 * gamma_mean_M1^x
AIC_M1 <- -2 * sum(dnorm(Y, mean_normal_M1, sigma2_mean_M1^0.5, log=T)) + 2 *4

beta0_mean_M2 <- mean(M2_post_samples$beta_0_samples)
beta1_mean_M2 <- mean(M2_post_samples$beta_1_samples)
gamma_mean_M2 <- mean(M2_post_samples$gamma_samples)
sigma2_mean_M2 <- mean(M2_post_samples$sigma2_samples)

mean_normal_M2 <- beta0_mean_M2 - beta1_mean_M2 * gamma_mean_M2^x
AIC_M2 <- -2 * sum(dgamma(Y, shape = mean_normal_M2 / sigma2_mean_M2, rate = 1 /
 ↪ sigma2_mean_M2, log = T)) + 2 * 4
AIC_M1
AIC_M2
#data_plot_mcmc_base <- data.frame(iterations = 1:
 ↪length(M1_v2_post_samples$beta_0_samples), beta0 =␣
 ↪M1_v2_post_samples$beta_0_samples, beta1 =␣
 ↪M1_v2_post_samples$beta_1_samples, gamma = M1_v2_post_samples$gamma_samples,␣
 ↪sigma2 = M1_v2_post_samples$sigma2_samples)
#AIC_M1 =
```

-1.92999798035484

16.0682277849034