STAT 578 - Fall 2019 - Assignment 4

Frederick (Eric) Ellwanger - fre2

November 1, 2019

Exercise 1

According to one version of Moore's law, the number of transistors on a state-of-the-art computer microprocessor roughly doubles every two years:

$$T \approx C2^{A/2}$$

(1)(a) Consider the natural logarithm of the number of transistors: log T.

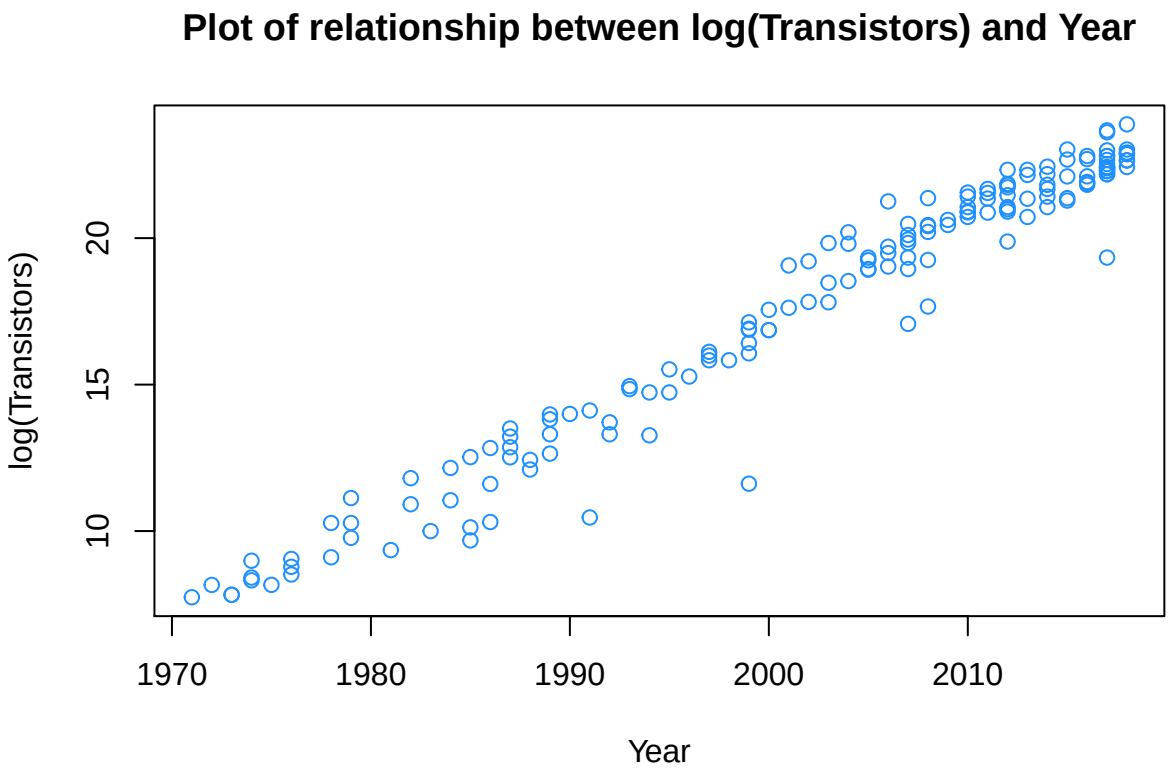
(1)(a)(i) Demonstrate that, according to Moore's law, log T should roughly follow a simple linear regression on A. Also, what should be the value of the coeffcient of A?

$log(T) \approx log(C2^{A/2})$
$log(T) \approx log(C) + log(2^{A/2})$
$log(T) \approx log(C) + A/2 * log(2)$

This demonstrates that log(T) roughly follows a simple linear regression of the form:
$log(T) \approx \beta_1 + \beta_2 A$ **where** $\beta_1 = log(C)$ **and** $\beta_2 = \frac{log(2)}{2}$
The coefficient of A is $\frac{log(2)}{2} \approx$ **0.347**

(1)(a)(ii) Plot the data points as log transistor count versus year.

```
ml = read.csv("mooreslawdata.csv", header=TRUE)
plot(log(Transistors) ~ Year, data = ml,
     main = "Plot of relationship between log(Transistors) and Year", col = "dodgerblue")
```



(1)(b) Consider a normal-theory simple linear regression model of log transistor count on centered year of the form:

$$log(T_i)|\beta, \sigma^2, A_i \quad \sim \quad indep.N(\beta_1 + \beta_2(A_i - \bar{A}), \sigma^2) \qquad i = 1, ..., 161$$

where $\bar{A}$ is the avergae of A_i over all observations. Of course, Moore's law specifies a particular value for β_2, but your initial model will not assume this. Use independent priors

$$\beta_1, \beta_2 \quad \sim \quad iidN(0, 1000^2)$$

$$\sigma^2 \quad \sim \quad Inv - gamma(0.001, 0.001)$$

(1)(b)(i) List an appropriate JAGS model

```
ml$logT = log(ml$Transistors)
ml$Ycent = ml$Year - mean(ml$Year)
```

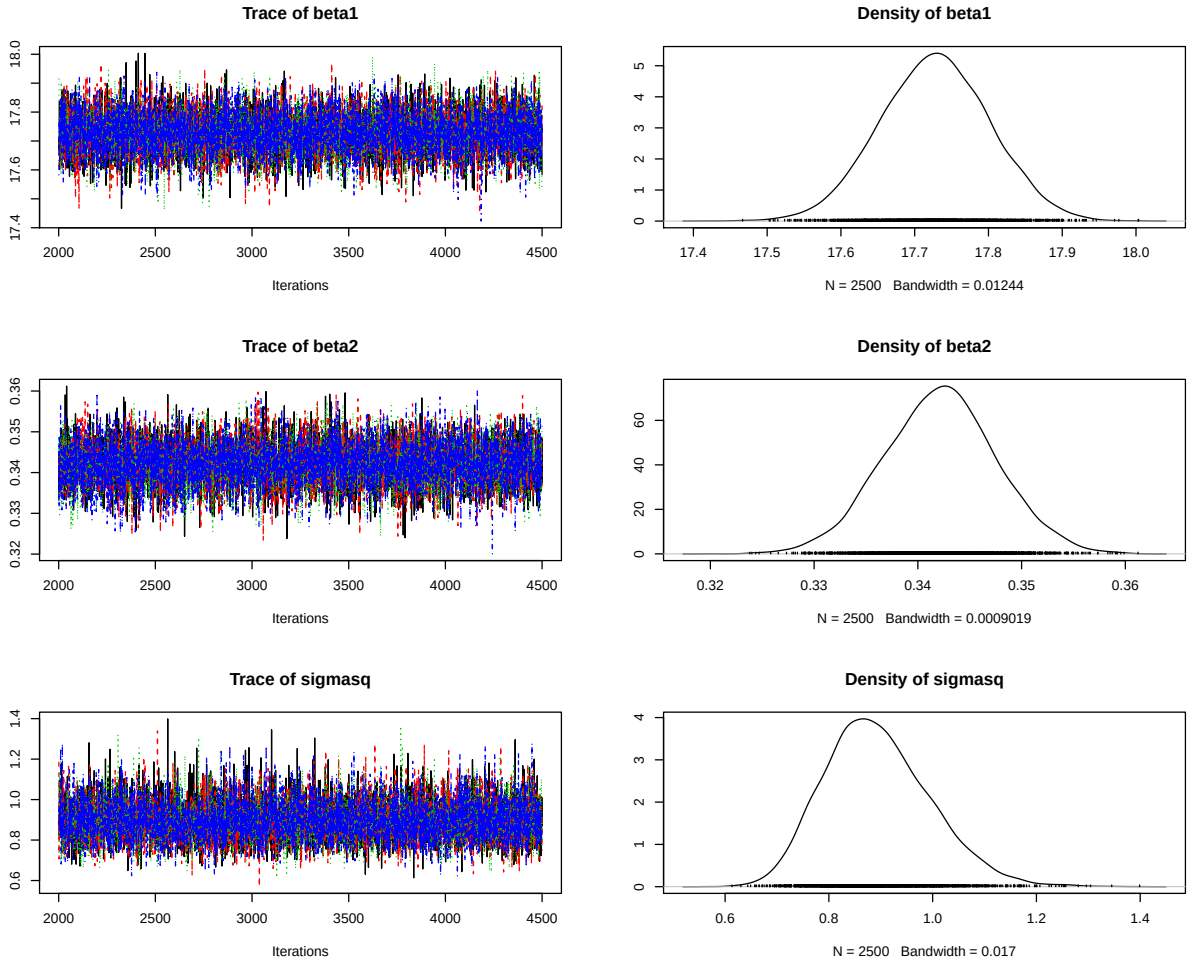
```
model {
   for(i in 1:length(logT)) {
      logT[i] ~ dnorm(beta1 + beta2*Ycent[i], sigmasqinv)
   }

   beta1 ~ dnorm(0, (1/1000^2))
   beta2 ~ dnorm(0, (1/1000^2))
```

```
   sigmasqinv ~ dgamma(0.001, 0.001)

   sigmasq = 1/sigmasqinv
   }
}
```

```
library(rjags)

#Set overdispersed starting points
inits = list(list(beta1 = 1700, beta2 = 1700, sigmasqinv = 100),
             list(beta1 = 1700, beta2 = -1700, sigmasqinv = 0.001),
             list(beta1 = -1700, beta2 = 1700, sigmasqinv = 0.001),
             list(beta1 = -1700, beta2 = -1700, sigmasqinv = 100))

#Setup model
chains = 4
m1 = jags.model("transistor.bug", ml, inits, n.chains = chains)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 161
##    Unobserved stochastic nodes: 3
##    Total graph size: 422
##
## Initializing model
```

```
#Run burn-in
update(m1, 2000)

#Run model
iters = 2500
x1 = coda.samples(m1, c("beta1", "beta2", "sigmasq"), n.iter = iters)

#Check for convergence
plot(x1, smooth = FALSE)
```



None of the parameter graphs show any concerns with convergence

```
#Another check for problems with convergance
gelman.diag(x1, autoburnin = FALSE)
```

```
## Potential scale reduction factors:
##
##         Point est. Upper C.I.
## beta1            1          1
## beta2            1          1
## sigmasq          1          1
##
## Multivariate psrf
##
## 1
```

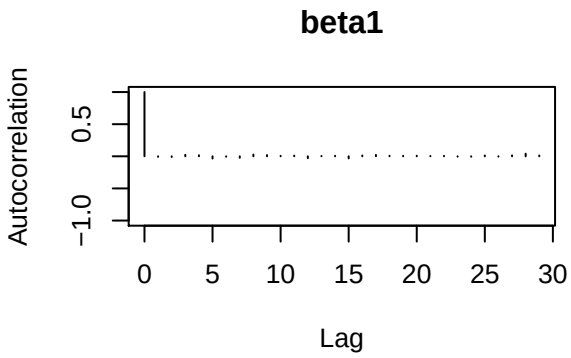
Gelman-Rubin statistics do not show any issues with convergance

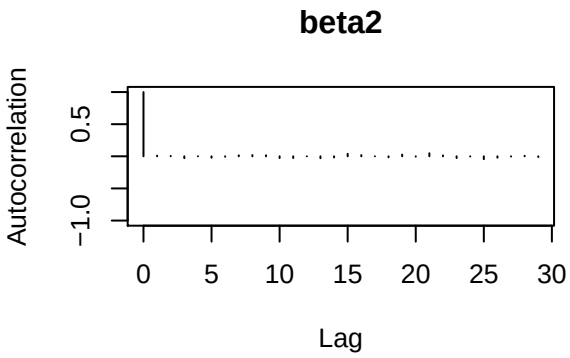
```
#Another check for convergance
effectiveSize(x1)
```

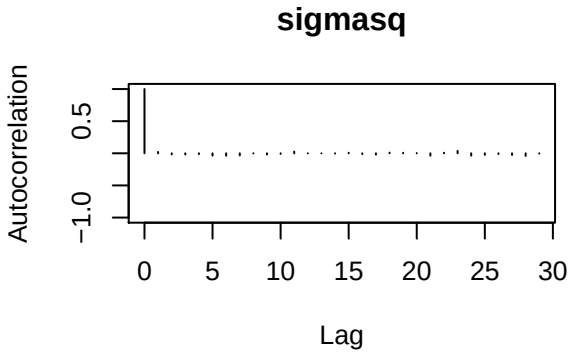
```
##   beta1   beta2 sigmasq
##   10000   10000    9856
```

The sample sizes are adequate

```
#Another check for convergance (just first chain )
autocorr.plot(x1[[1]])
```







Autocorrelation plots do not show any issues with mixing

There does not appear to be any issues with convergance

(1)(b)(ii) List the `coda` summary of your results for β_1, β_2, and σ^2

```
summary(x1)
```

```
##
## Iterations = 2001:4500
## Thinning interval = 1
## Number of chains = 4
## Sample size per chain = 2500
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##           Mean      SD  Naive SE Time-series SE
## beta1   17.728 0.07404 0.0007404      0.0007404
## beta2    0.342 0.00541 0.0000541      0.0000541
## sigmasq  0.894 0.10178 0.0010178      0.0010256
##
## 2. Quantiles for each variable:
##
```

```
##           2.5%    25%    50%    75%  97.5%
## beta1   17.583 17.678 17.728 17.778 17.872
## beta2    0.331  0.339  0.342  0.346  0.353
## sigmasq  0.718  0.822  0.885  0.958  1.110
```

(1)(b)(iii)Give the approximate posterior mean and 95% posterior credible interval for the slope. Does the interval contain the value you determined in part (a), in accordance with Moore's law?

```
summary(x1)$statistics['beta2','Mean']
```

```
## [1] 0.342
```

The approximate posterior mean for the slope is 0.342

```
summary(x1)$quantiles['beta2',c('2.5%', '97.5%')]
```

```
##  2.5% 97.5%
## 0.331 0.353
```

The approximate 95% posterior credible interval for the slope is (0.331, 0.353)

This interval does contain the value calculted in part (a) : 0.347

(1)(b)(iv)Give the approximate posterior mean and 95% posterior credible interval for the intercept.

```
summary(x1)$statistics['beta1','Mean']
```

```
## [1] 17.7
```

The approximate posterior mean for the intercept is 17.728

```
summary(x1)$quantiles['beta1',c('2.5%', '97.5%')]
```

```
##  2.5% 97.5%
##  17.6  17.9
```

The approximate 95% posterior credible interval for the intercept is (17.583, 17.872)

(1)(c) Consider the model of the previous part. You will use it to predict the transistor count on a microprocessor introduced in 2020, and also (just for fun) to see if it extrapolates back to the invention of the transistor.

(1)(c)(i) List a modifed JAGS model appropriate for answering the subparts below.

```
model {
   for(i in 1:length(Transistors)) {
      log(Transistors[i]) ~ dnorm(beta1 + beta2*Ycent[i], sigmasqinv)
   }
   Tnew ~ dnorm(beta1 + beta2*YNewcent, sigmasqinv)

   beta1 ~ dnorm(0, (1/1000^2))
   beta2 ~ dnorm(0, (1/1000^2))
   sigmasqinv ~ dgamma(0.001, 0.001)

   sigmasq = 1/sigmasqinv

   A0 = mean(Year) - beta1/beta2
```

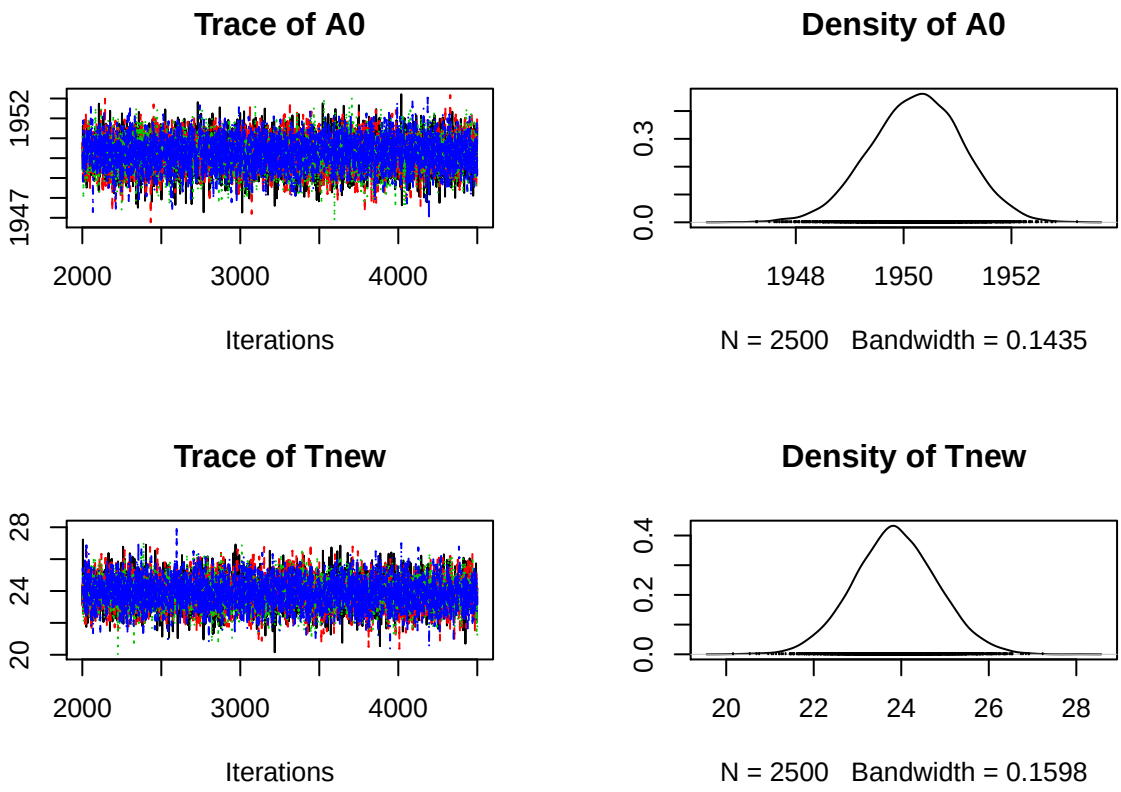
```
}
```

Now run your model. Make sure to use multiple chains with overdispersed starting points, check convergence, and monitor parameters for at least 2000 iterations (per chain) after burn-in.

```
#Set overdispersed starting points
inits = list(list(beta1 = 1700, beta2 = 1700, sigmasqinv = 100),
             list(beta1 = 1700, beta2 = -1700, sigmasqinv = 0.001),
             list(beta1 = -1700, beta2 = 1700, sigmasqinv = 0.001),
             list(beta1 = -1700, beta2 = -1700, sigmasqinv = 100))

#Setup model
Newcent = 2020 - mean(ml$Year)
m2 = jags.model("transistormod.bug", as.list(c(ml, YNewcent = Newcent)),
                inits, n.chains = 4)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 161
##    Unobserved stochastic nodes: 4
##    Total graph size: 594
##
## Initializing model
```

```
#Run burn-in
update(m2, 2000)

#Run model
x2 = coda.samples(m2, c("Tnew", "A0"), n.iter = 2500)

#check for convergance
plot(x2, smooth = FALSE)
```



None of the parameter graphs show any concerns with convergence

```
#Another check for problems with convergance
gelman.diag(x2, autoburnin = FALSE)
```

```
## Potential scale reduction factors:
##
##      Point est. Upper C.I.
## A0            1          1
## Tnew          1          1
##
## Multivariate psrf
##
## 1
```

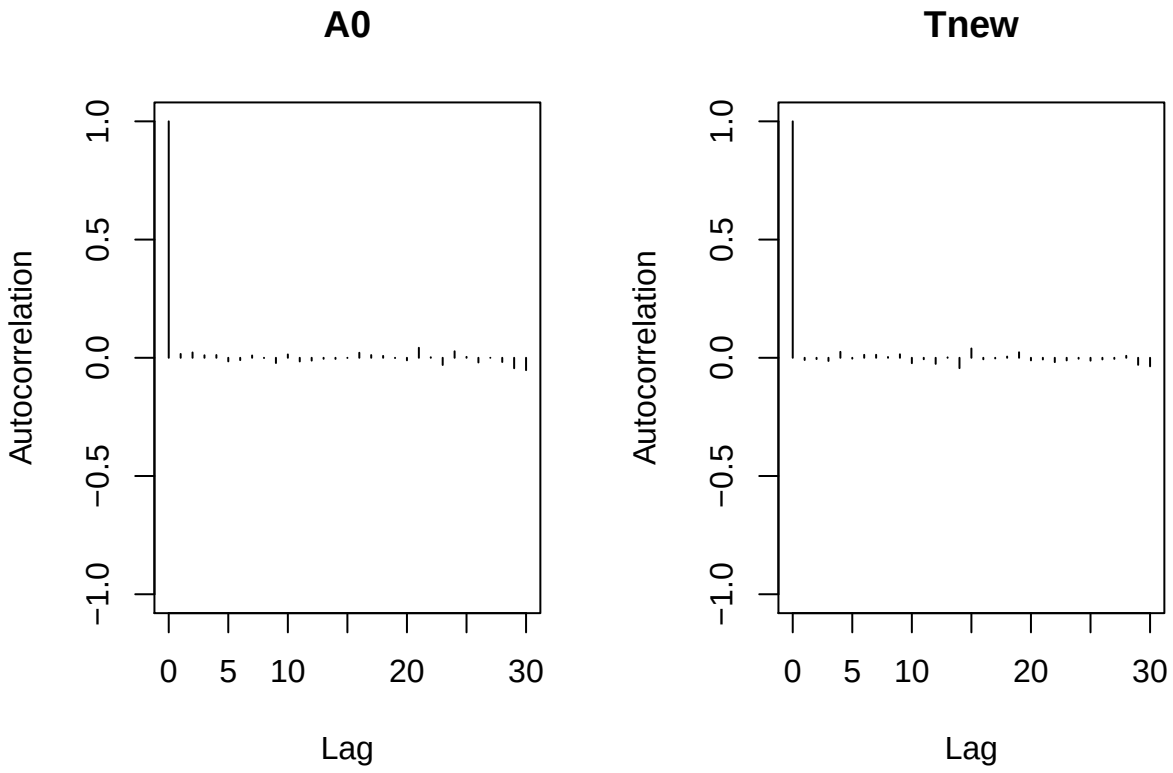
Gelman-Rubin statistics do not show any issues with convergance

```
#Another check for convergance
effectiveSize(x2)
```

```
##    A0  Tnew
##  9789 10000
```

The sample sizes are adequate

```
#Another check for convergance (just first chain )
autocorr.plot(x2[[1]])
```



Autocorrelation plots do not show any issues with mixing

There does not appear to be any issues with convergance

(1)(c)(ii) List the coda summary you will use to help answer the subparts below.

```
summary(x2)
```

```
##
## Iterations = 2001:4500
## Thinning interval = 1
## Number of chains = 4
## Sample size per chain = 2500
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##        Mean    SD Naive SE Time-series SE
## A0   1950.2 0.854  0.00854        0.00864
## Tnew   23.9 0.960  0.00960        0.00960
##
## 2. Quantiles for each variable:
##
##      2.5%    25%    50%    75%  97.5%
## A0   1949 1949.7 1950.3 1950.8 1951.9
## Tnew   22   23.2   23.9   24.5   25.8
```

(1)(c)(iii) Give an approximate 95% posterior predictive interval for the transistor count, in billions, on a microprocessor introduced in the year 2020. (Note: This is for the count, NOT the log count.)

```
exp(summary(x2)$quantiles['Tnew',c('2.5%', '97.5%')])/1000000000
```

```
##   2.5%  97.5%
##   3.51 157.29
```

The approximate 95% posterior predictive interval for the transistor count in billions, on a m icropocessor introduced in the year 2020 is (3.515, 157.289)

(1)(c)(iv) Explain why the model suggests that the transistor was invented in the year:

$$\bar{A} - \frac{\beta_1}{\beta_2}$$

To find the year the model predicts the transistor was invented, we need to find where the transistor count is 0

$$y = 0 = \beta_1 + \beta_2 * (A - \bar{A})$$

$$-\frac{\beta_1}{\beta_2} = A - \bar{A}$$

$$A = \bar{A} - \frac{\beta_1}{\beta_2}$$

This explains how the model suggests the transistor was invented in the year $\bar{A} - \frac{\beta_1}{\beta_2}$

give an approximate 95% posterior interval for this quantity. (You may compare this to the actual year in which the transistor was invented.)

```
summary(x2)$quantiles['A0',c('2.5%', '97.5%')]
```

```
##  2.5% 97.5%
##  1949  1952
```

The approximate 95% posterior interval for this quantity is (1949, 1952)

This compares very close to the actual year of 1947 (really December 23, 1947, so you could almost call that 1948)

(1)(d) One way to check for evidence of outliers is a posterior predictive p-value based on test quantity

$$T(y, X, \theta) = \max_i |\frac{\epsilon_i}{\sigma}|$$

where ϵ_i is the error for observation i. The larger this quantity is, the more we should suspect the existence of an outlier.

(1)(d)(i) Show R code for computing the simulated error vectors ϵ (as rows of a matrix).

```r
#Get matrixes/vectors from simulated MCMC model
betas.sim = as.matrix(x1)[, 1:2]
sigma.2.sim = as.matrix(x1)[, 3]

#Create X matrix with first column as 1's
X = as.matrix(cbind(rep(1, nrow(ml)), ml$Ycent))

nsims = chains*iters

#Create empty matrix to store error vectors
error.sim = matrix(NA, nsims, nrow(ml))

#Calculate simulated error vectors
for (s in 1:nsims){
       error.sim[s, ] = log(ml$Transistors) - X %*% cbind(betas.sim[s, ])
}
```

(1)(d)(ii) Show R code for computing simulated replicate error vectors ϵ^{rep} (as rows of a matrix), which are the error vectors for the replicate response vectors y^{rep}.

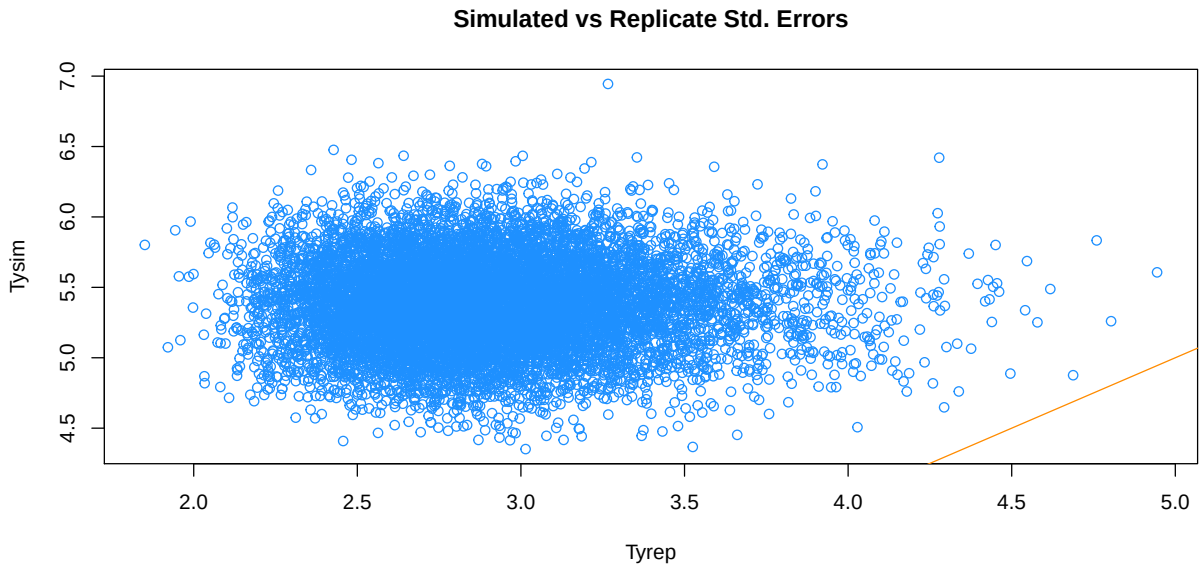
```r
#Replicated errors
error.rep = matrix(rnorm(nsims*nrow(ml), mean = 0, sd = sqrt(sigma.2.sim)),
                   nsims, nrow(ml))
```

(1)(d)(iii) Show R code for computing the simulated values of $T(y, X, \theta)$ and the simulated values of $T(y^{rep}, X, \theta)$

```r
#Simulated values of T(y, X, theta)
Tysim = apply(abs(error.sim/sqrt(sigma.2.sim)), 1, max)

#Simulated values of (Yrep, X, theta)
Tyrep = apply(abs(error.rep)/sqrt(sigma.2.sim), 1, max)
```

(1)(d)(iv) Plot the simulated values of $T(y^{rep}, X, \theta)$ versus those of $T(y, X, \theta)$, with a reference line indicating where $T(y^{rep}, X, \theta) = T(y, X, \theta)$.

```r
plot(Tyrep, Tysim, col = "dodgerblue", main = "Simulated vs Replicate Std. Errors")
abline(0, 1, col = "darkorange")
```



(1)(d)(v) Compute the approximate posterior predictive p-value, and make an appropriate conclusion based on it. (Is there evidence for an outlier?)

```
mean(Tyrep >=Tysim)
```

```
## [1] 0
```

The approximate posterior predictive p-value is 0

The test quantity and plot seem to indicate that there may be an issue with outliers

(1)(d)(vi) Name the microprocessor that appears to be the most extreme outlier.

```
#find largest error in each yi
p = apply(abs(error.sim/sqrt(sigma.2.sim)), 1, which.max)

#Find most common max error
y = table(p)
(proc = as.character(ml$Processor[as.numeric(names(y)[which(y==max(y))])]))
```

```
## [1] "ARM 9TDMI"
```

The micrproccesor that appears to be the most extreme outlier is the ARM 9TDMI