A4

Shyam Shah

30/03/2021

1 (a)

First show that $log(C)$ roughly follows a simple linear regression.

$$C \approx \gamma 2^{A/2}$$

$$log(C) \approx log(\gamma 2^{A/2})$$

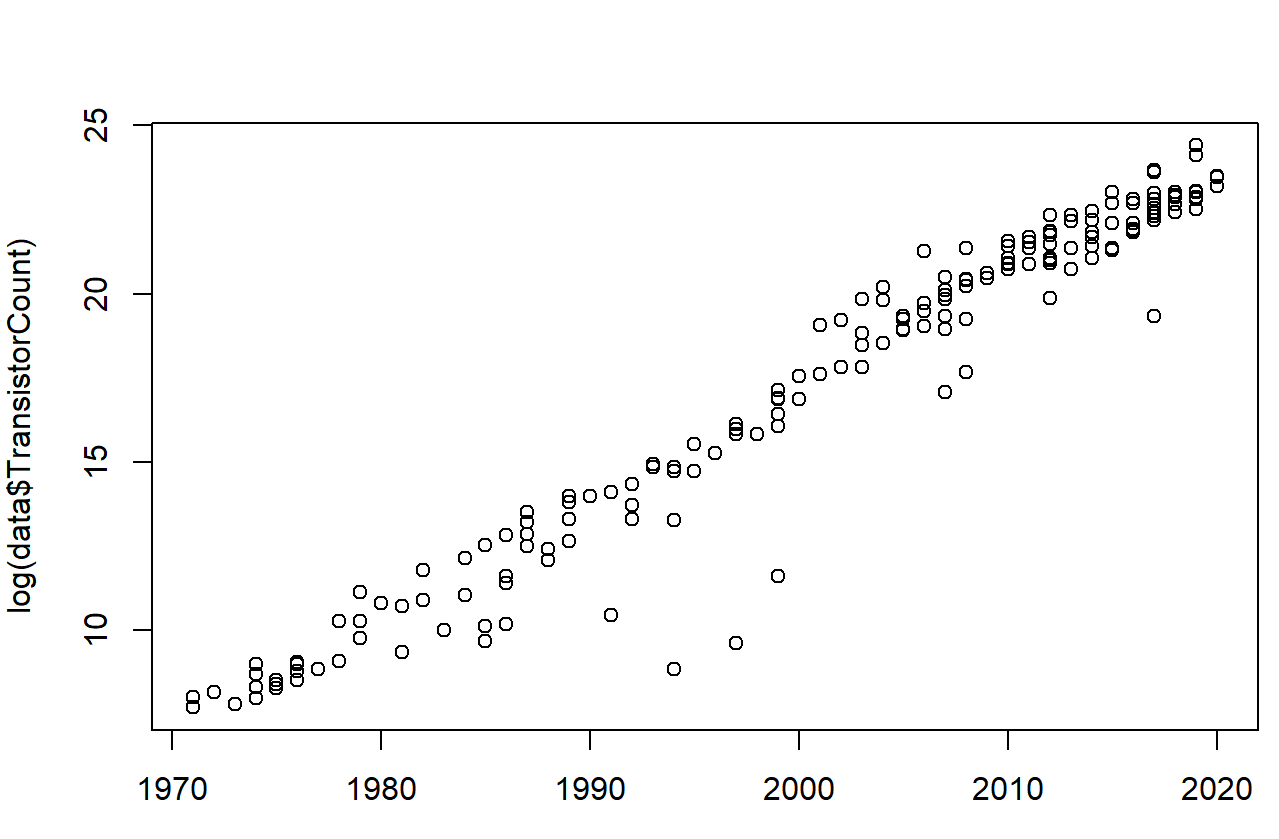
$$log(C) \approx log(\gamma) + log(2^{A/2})$$

$$log(C) \approx log(\gamma) + \frac{A}{2} log(2)$$

This shows that $log(c)$ roughly follows a simple linear regression on A with $log(\gamma)$ is the intercept and the coefficient for A ($log(2)/2$) is the slope (ie. the measure of the relation between A and C.

1 (b)

```
data <- read.csv("./mooreslawdata.csv", header=TRUE)
plot(data$Year, log(data$TransistorCount)) # plot the data
```



1 (c) (i)

This is what the initial model looks like:

```
model {
  for (j in 1:length(TransistorCount)) {
    TransistorCount[j] ~ dnorm(beta1+beta2*(Yearcentered[j]), sigmasqinv)
  }

  beta1 ~ dnorm(0, 0.000001)
  beta2 ~ dnorm(0, 0.000001)
  sigmasqinv ~ dgamma(0.001, 0.001)

  sigmasq <- 1/sigmasqinv
}
```

```
#prepare data for the model
d1 <- list(TransistorCount = log(data$TransistorCount),
           Yearcentered = data$Year - mean(data$Year))

#prepare chains for the model too
inits1 <- list(list(beta1=1000, beta2=1000, sigmasqinv=0.1, .RNG.name="base::Wichmann-Hill", .RNG.seed=123),
              list(beta1=1000, beta2=1000, sigmasqinv=0.00001, .RNG.name="base::Wichmann-Hill", .RNG.seed=124),
              list(beta1=1000, beta2=-1000, sigmasqinv=0.1, .RNG.name="base::Wichmann-Hill", .RNG.seed=125),
              list(beta1=1000, beta2=-1000, sigmasqinv=0.00001, .RNG.name="base::Wichmann-Hill", .RNG.seed=126))

library(rjags)
```

```
## Loading required package: coda
```

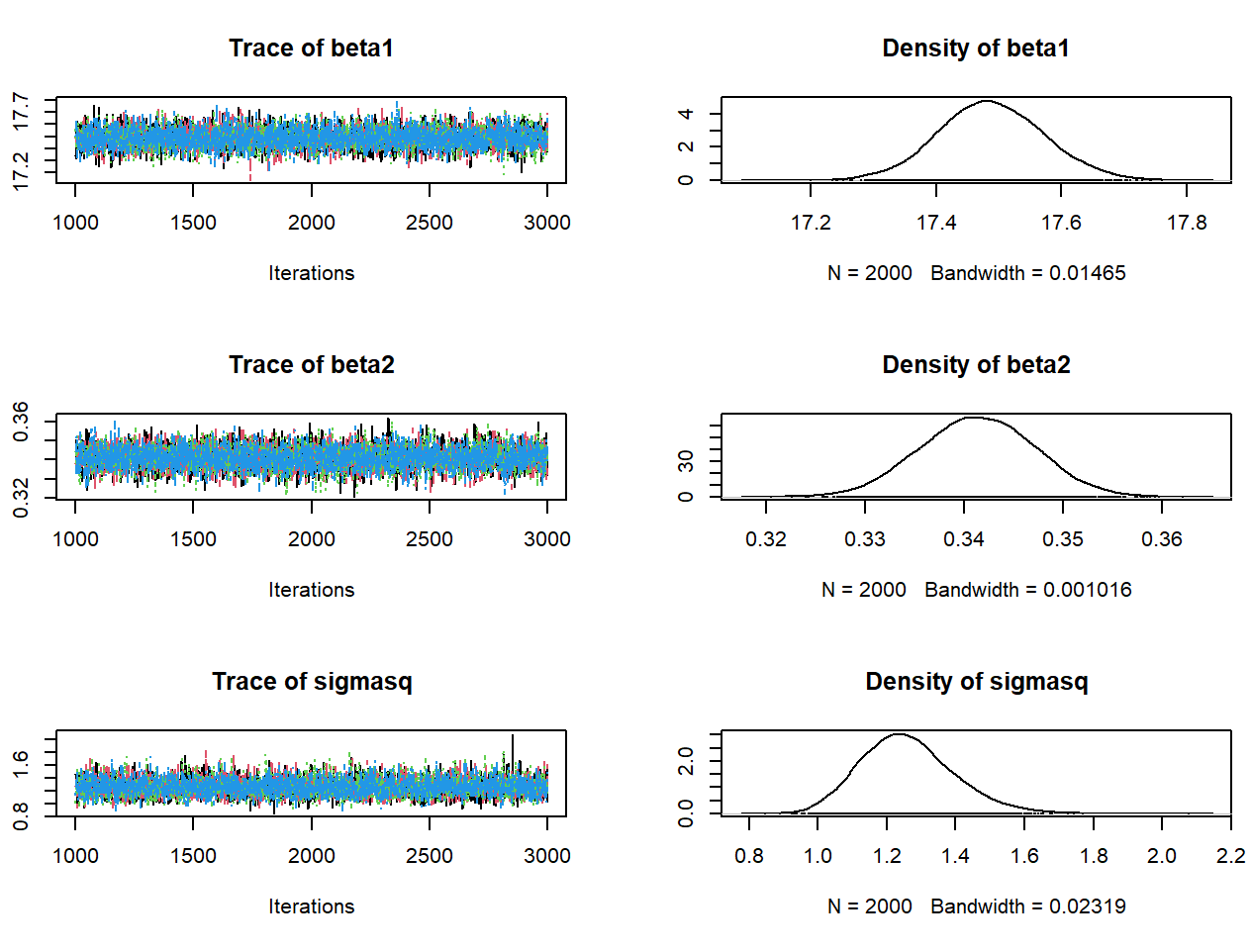
```
## Linked to JAGS 4.3.0
```

```
## Loaded modules: basemod,bugs
```

```
#create the model with 1000 iterations for adaption
m1 <- jags.model("./mooresic.bug", d1, inits1, n.chains=4, n.adapt=1000)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 182
##    Unobserved stochastic nodes: 3
##    Total graph size: 474
##
## Initializing model
```

```
#discard results of another 1000 iterations for burn in
update(m1, 1000)

#check results of beta1, beta2, and sigmasq
x1 <- coda.samples(m1, c("beta1", "beta2", "sigmasq"), n.iter=2000)

#plot results to check for convergence
plot (x1, smooth=FALSE)
```




```
#check Gelman-Rubin statistic
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
```

Based on this, it looks like there is convergence.

1 (c) (ii)

```
summary(x1)
```

```
##
## Iterations = 1001:3000
## Thinning interval = 1
## Number of chains = 4
## Sample size per chain = 2000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##            Mean       SD  Naive SE Time-series SE
## beta1   17.4878 0.083373 9.321e-04      9.459e-04
## beta2    0.3413 0.005785 6.467e-05      6.388e-05
## sigmasq  1.2595 0.134835 1.507e-03      1.517e-03
##
## 2. Quantiles for each variable:
##
##            2.5%     25%     50%     75%   97.5%
## beta1   17.3261 17.4310 17.4857 17.5441 17.6562
## beta2    0.3301  0.3374  0.3413  0.3453  0.3526
## sigmasq  1.0231  1.1648  1.2496  1.3417  1.5531
```

1 (c) (iii)

The approximate posterior mean for the slope is 0.3413 and the 95% posterior interval is $(0.3301, 0.3526)$. The value determined in part (a) is $\frac{log(2)}{2} \approx 0.3466$, (log refers to the natural logarithm here) so the interval does contain this value.

1 (c) (iv)

The approximate posterior mean for the intercept is 17.4898 and the 95% posterior interval is $(17.3242, 17.6511)$.

1 (d)

```
# calculate Ai-Abar for the year 2022
2022-mean(data$Year)
```

```
## [1] 20.39011
```

```
# calculate Abar
mean(data$Year)
```

```
## [1] 2001.61
```

The modified JAGS model is below where TransistorPred is the prediction for year 2022, and InventedYear is the prediction for the year transistors were invented:

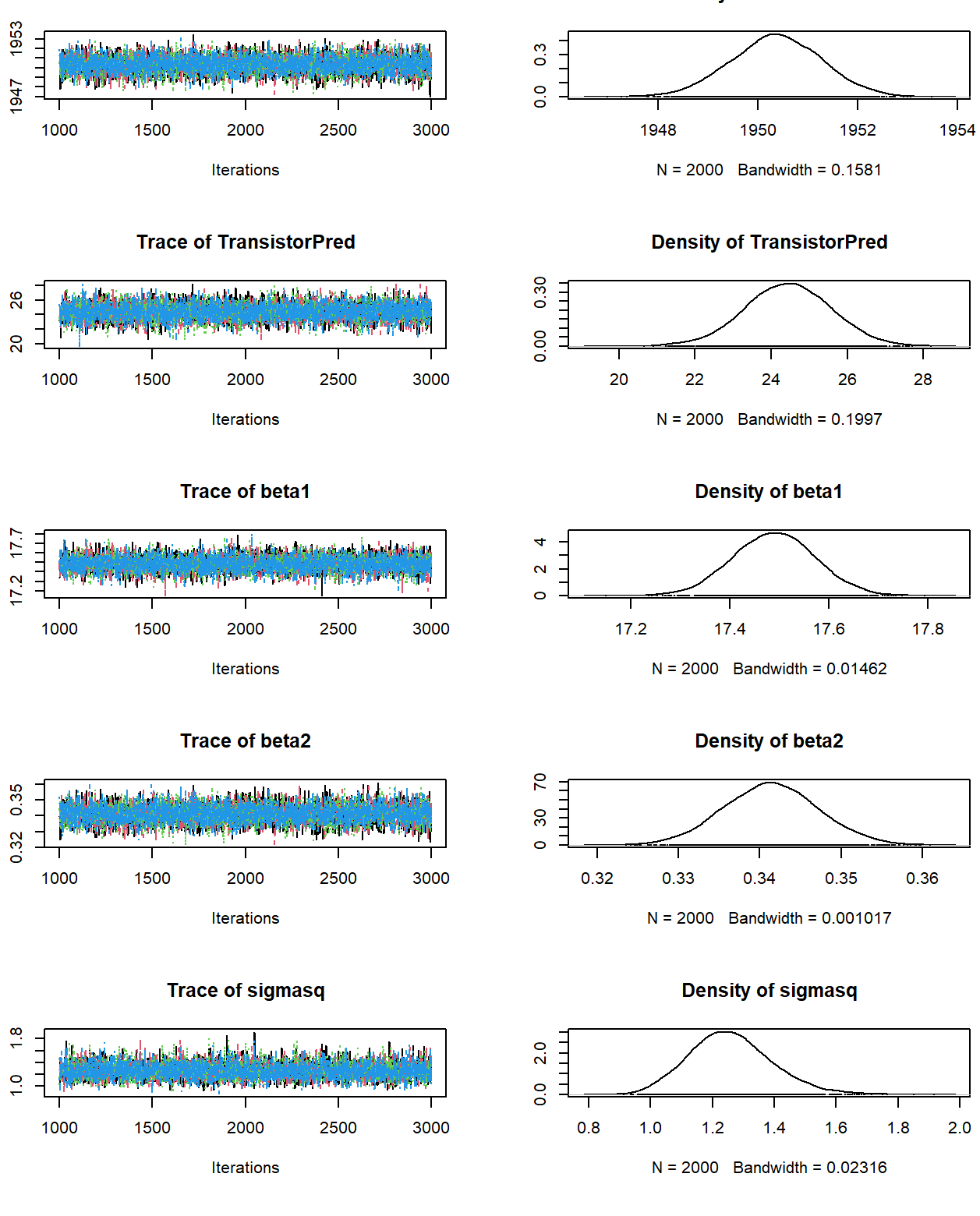
```
model {
  for (j in 1:length(TransistorCount)) {
    TransistorCount[j] ~ dnorm(beta1+beta2*(Yearcentered[j]), sigmasqinv)
  }

  beta1 ~ dnorm(0, 0.000001)
  beta2 ~ dnorm(0, 0.000001)
  sigmasqinv ~ dgamma(0.001/2, 0.001/2)

  TransistorPred ~ dnorm(beta1+beta2*(20.39011), sigmasqinv)
  InventedYear <- 2001.61-(beta1/beta2)


  sigmasq <- 1/sigmasqinv
}
```

And now to run the model and check for convergence:

```
library(rjags)

#create the model with 1000 iterations for adaption
m2 <- jags.model("./mooresid.bug", d1, inits1, n.chains=4, n.adapt=1000)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 182
##    Unobserved stochastic nodes: 4
##    Total graph size: 481
##
## Initializing model
```

```
#discard results of another 1000 iterations for burn in
update(m2, 1000)

#check results of beta1, beta2, and sigmasq
x2 <- coda.samples(m2, c("beta1", "beta2", "sigmasq", "TransistorPred", "InventedYear"), n.iter=2000)

#plot results to check for convergence
plot (x2, smooth=FALSE)
```




```
#check Gelman-Rubin statistic
gelman.diag(x2, autoburnin = FALSE)
```

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

Once again the graphs show chains sampling from the same areas so there is convergence. Also the Gelman-Rubin statistics are near 1.

1 (d) (ii)

```
summary(x2)
```

```
##
## Iterations = 1001:3000
## Thinning interval = 1
## Number of chains = 4
## Sample size per chain = 2000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                    Mean       SD  Naive SE Time-series SE
## InventedYear   1950.3632 0.899918 1.006e-02      1.006e-02
## TransistorPred   24.4376 1.136933 1.271e-02      1.326e-02
## beta1            17.4888 0.083216 9.304e-04      9.300e-04
## beta2             0.3414 0.005788 6.471e-05      6.471e-05
## sigmasq           1.2591 0.134443 1.503e-03      1.488e-03
##
## 2. Quantiles for each variable:
##
##                   2.5%       25%       50%       75%     97.5%
## InventedYear   1948.59 1949.7706 1950.3699 1950.9836 1952.1299
## TransistorPred   22.15   23.6738   24.4417   25.2101   26.6501
## beta1            17.33   17.4319   17.4892   17.5450   17.6525
## beta2             0.33    0.3374    0.3414    0.3452    0.3528
## sigmasq           1.02    1.1662    1.2520    1.3428    1.5444
```

1 d (iii)

Since we want the 95% central posterior predictive interval for the transistor count in billions, we can get this by taking the numbers from the summary above and converting them to the appropriate scale. So the interval is $(\frac{e^{22.15}}{10^9}, \frac{e^{26.6501}}{10^9})$ or $(4.165, 374.9656)$

1 d (iv)

The model we have can be written as:

$$log(C) = \beta_1 + \beta_2(A_i - \bar{A})$$

If we want to find out what year the transistor was created, we can set $C = 1$ and so $log(C) = 0$ and solve for A_i

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

$$-\beta_1 = \beta_2(A_i - \bar{A})$$

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

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

The results of 1dii show that the 95% posterior predictive interval for this value is $(1948.59, 1952.1299)$ and the actual year it was invented was 1947.

1 (e) (i)

```
library (MASS)

#use classic linear regression to get X values
classic.mod <- lm(TransistorCount ~ Yearcentered, data=d1)
X <- model.matrix(classic.mod)

#gather posterior simulation
Nsim <- 2000

post.sigma.2.sim <- as.matrix(x1)[,"sigmasq"]
post.beta1.sim <- as.matrix(x1)[,"beta1"]
post.beta2.sim <- as.matrix(x1)[,"beta2"]


#now create the simulated standardized error vectors
error.std.sim <- matrix(NA, Nsim, nrow(data))
for (s in 1:Nsim)
  error.std.sim[s,] <- (log(data$TransistorCount) - X %*% rbind(post.beta1.sim[s], post.beta2.sim[s]))/sqrt(post.
sigma.2.sim[s])
```

The simulated standardized error vector is error.std.sim.

1 (e) (ii)

```
#first compute the replicates
n<-nrow(data)
yreps <- matrix(NA, 2000, n)
for (s in 1:2000)
  yreps[s,] <- rnorm(n, X %*% rbind(post.beta1.sim[s], post.beta2.sim[s]), sqrt(post.sigma.2.sim[s]))

#then compute their standardized error vectors
error.std.sim.rep <- matrix(NA, Nsim, nrow(data))
for (s in 1:Nsim)
  error.std.sim.rep[s,] <- (yreps[s,] - X %*% rbind(post.beta1.sim[s], post.beta2.sim[s]))/sqrt(post.sigma.2.sim
[s])
```

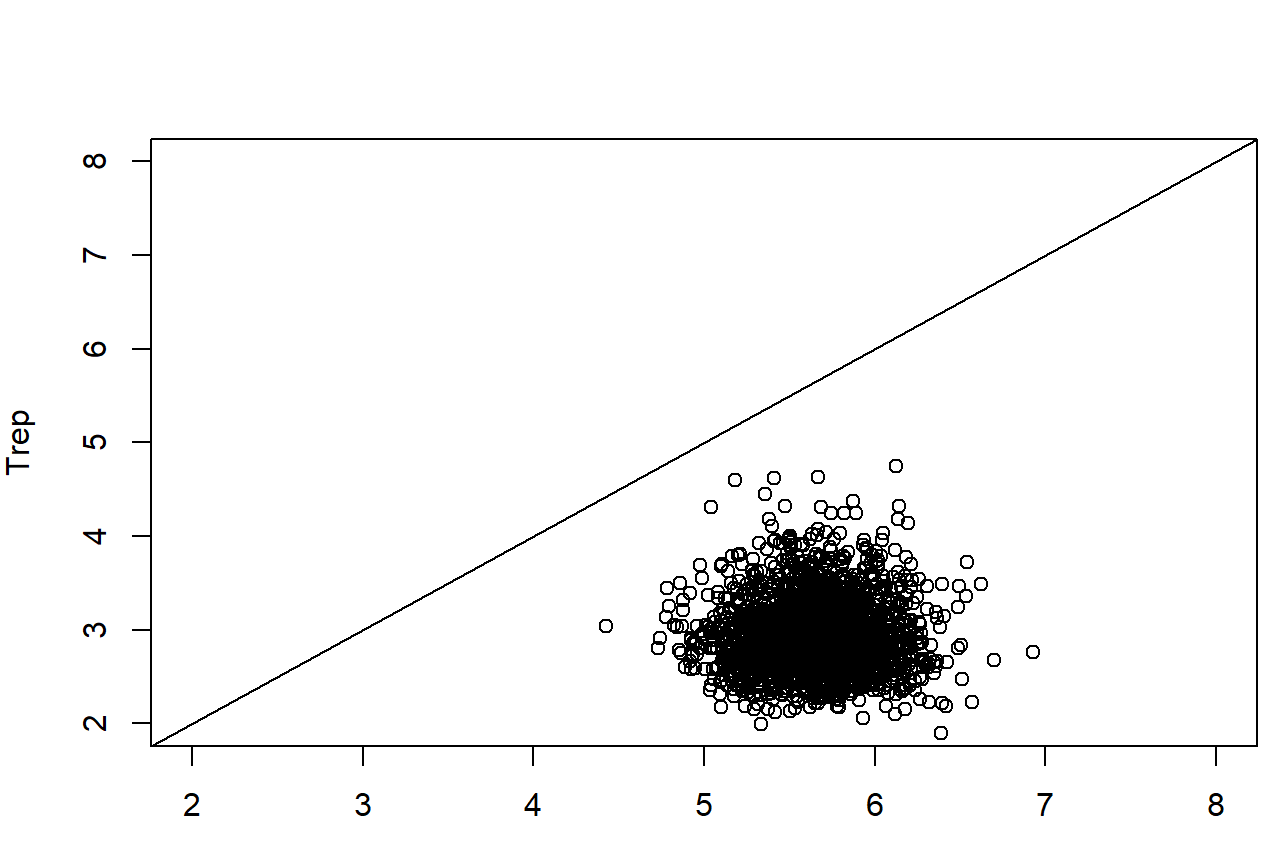
The replicate simulated standardized error vector is error.std.sim.rep.

1 (e) (iii)

```
#first compute T(y, X, theta)
T <- apply (abs(error.std.sim), 1, max)

#then compute T(yrep, X, theta)
Trep <- apply (abs(error.std.sim.rep), 1, max)
```

1 (e) (iv)

```
#plot T versus Trep
plot(T,Trep, xlim=c(2,8), ylim=c(2,8))
abline(coef=c(0,1))
```



1 (e) (v)

```
mean(apply(yreps,1,max)>=max(d1$TransistorCount))
```

```
## [1] 0.988
```

The approximate posterior predictive p-value is very close to 1 so there is evidence of outliers.

1 (e) (vi)

```
#first get absolute value of difference between average replicated ys and observed ones
diffs<-abs(apply(yreps,2,mean)-d1$TransistorCount)
#then find the index for the largest average and look up which processor it was
data$Processor[which.max(diffs)]
```

```
## [1] "F21"
```

So the F21 processor appears to be the most extreme outlier.