

# HW06 Solutions

BJ

2022-03-31

```
set.seed(1234567890) ## to make a reproducible simulation result

library(arm)
library(ggplot2); theme_set(theme_bw())
library(gridExtra)

library(rstan)
options(mc.cores = parallel::detectCores())
rstan_options(auto_write = TRUE)

library(bayesplot)
library(shinystan)

library(DHARMa)

print_file <- function(filename) {
  x <- paste(scan(file=filename,
                  what="",
                  sep="\n",
                  blank.lines.skip = TRUE,
                  quiet=TRUE),
             collapse="\n")
  cat(x)
}
```

## Problem 1 – Chicks

The `ChickWeight` data frame is included with R, and you can access it with `data(ChickWeight)`. It has 578 rows and 4 columns from an experiment on the effect of diet on early growth of chicks. The variables (Columns) are:

- weight: a numeric vector giving the body weight of the chick (gm).
- Time: a numeric vector giving the number of days since birth when the measurement was made.
- Chick: an ordered factor with levels  $18 < \dots < 48$  giving a unique identifier for the chick. The ordering of the levels groups chicks on the same diet together and orders them according to their final weight (lightest to heaviest) within diet.
- Diet: a factor with levels 1, ..., 4 indicating which experimental diet the chick received.

For most of this exercise we will be working with the model

$$\begin{aligned}\log(\text{weight}_i) &= \alpha_{0j[i]} + \alpha_{1j[i]} \cdot \text{Time}_i + \epsilon_i \\ \alpha_{0j} &= \beta_0 + \eta_{0j} \\ \alpha_{1j} &= \beta_1 + \eta_{1j} \\ \epsilon_i &\stackrel{iid}{\sim} N(0, \sigma^2) \\ \eta_{0j} &\stackrel{iid}{\sim} N(0, \tau_0^2) \\ \eta_{1j} &\stackrel{iid}{\sim} N(0, \tau_1^2),\end{aligned}$$

where  $i$  is the observation number and  $j$  is the Chick number<sup>1</sup>. You will have to convert `Chick` from a factor variable to a numerical variable, to work with `stan()`.

## Problem 1(a)

In the same folder as this assignment sheet there is a file `chicks.stan` with most of the program already written, but the model section left blank. Fill in the blank model section<sup>2</sup> and get the program to run using `library(rstan)`, taking at least 500 MCMC steps per chain<sup>3</sup>, and accepting the other `stan()` defaults, so that you end up with at least 1000 “good” MCMC samples from the four Markov chains produced. Turn in:

- The completed stan program `chicks.stan`.
- A printout of the fitted stan object but including only the lines for  $\beta_0$ ,  $\beta_1$ ,  $\tau_0$ ,  $\tau_1$ , and  $\sigma$ .

(You are free to use `lmer()` to estimate the corresponding multi-level model, to make sure you are on the right track, but please don't turn in any `lmer()` code or output for this part of the exercise.)

```
data(ChickWeight)
```

```
## str(ChickWeight)
```

Since the Level 1 mean  $\mu[i] \leftarrow a0[\text{Chick}[i]] + a1[\text{Chick}[i]] \cdot \text{Time}[i]$  and the random intercept  $a0[j] \leftarrow b0 + \eta_{0j}$  and slope  $a1[j] \leftarrow b1 + \eta_{1j}$  are already defined in the *transformed parameters* block, all we need to do in the *model* block is specify

- Level 1: distribution (likelihood) for  $y$ :

```
for (i in 1:N) {
  y[i] ~ normal(mu[i],sigma);
}
```

- Level 2: distributions of the  $\eta$ 's:

```
for (j in 1:J) {
  eta0[j] ~ normal(0,tau0);
  eta1[j] ~ normal(0,tau1);
}
```

- Priors for all the free parameters (fixed effects and sd's):

```
b0 ~ normal(0,1e6);
b1 ~ normal(0,1e6);
sigma ~ uniform(0,50);
tau0 ~ uniform(0,50);
tau1 ~ uniform(0,50);
```

<sup>1</sup>Also note that in this model we are just letting  $\rho_{\eta_0, \eta_1} = 0$ , to make the `stan()` coding less complicated.

<sup>2</sup>You should not have to change any of the code already in the file; just add to it. However if it is easier for you to change some of the code in the file, that is fine too.

<sup>3</sup>If you need to take more steps to get convergence to the stationary distribution, that is fine.

*The final, complete stan program looks like this:*

```
print_file("chicks.stan")
```

```
## data {
##   int N; // # observations
##   int J; // # chicks
##   real weight[N];
##   real Time[N];
##   int Chick[N]; // needs to be integers, not a factor variable
## }
##
## transformed data {
##   real y[N];
##   y <- log(weight);
## }
##
## parameters{
##   real b0;
##   real b1;
##   real eta0[J];
##   real eta1[J];
##   real<lower=0> sigma;
##   real<lower=0> tau0;
##   real<lower=0> tau1;
## }
##
## transformed parameters {
##   real mu[N];
##   real a0[J];
##   real a1[J];
##   for (j in 1:J) {
##     a0[j] <- b0 + eta0[j];
##     a1[j] <- b1 + eta1[j];
##   }
##   for (i in 1:N) {
##     mu[i] <- a0[Chick[i]] + a1[Chick[i]]*Time[i];
##   }
## }
##
## model {
##   // Level 1
##   for (i in 1:N) {
##     y[i] ~ normal(mu[i],sigma) ;
##   }
##   // Level 2
##   for (j in 1:J) {
##     eta0[j] ~ normal(0,tau0);
##     eta1[j] ~ normal(0,tau1);
##   }
##   // Prior distributions
##   b0 ~ normal(0,1e6);
##   b1 ~ normal(0,1e6);
##   sigma ~ uniform(0,50);
```

```
## tau0 ~ uniform(0,50);
## tau1 ~ uniform(0,50);
## }
##
## generated quantities {
##   real yrep[N];
##   for (i in 1:N) {
##     yrep[i] <- normal_rng(mu[i],sigma);
##   }
## }
```

*Examining the data block,*

```
data {
  int N; // # observations
  int J; // # chicks
  real weight[N];
  real Time[N];
  int Chick[N]; // needs to be integers, not a factor variable
}
```

*all of the variables are already in the ChickWeight data frame, except for N, J and a version of Chick that is an integer, not a factor, variable. Here is code to run the model:*

```
attach(ChickWeight)

chicknum <- as.numeric(Chick)

chickdata <- list(N=length(chicknum),
                 J=length(unique(chicknum)),
                 weight=weight,
                 Time=Time,
                 Chick=chicknum)

detach()

chickmodel <- stan(file="chicks.stan",data=chickdata,chains=0)

chickresult <- stan(fit=chickmodel,data=chickdata,iter=500)
```

*and here is the printout of the results, for  $\beta_0$ ,  $\beta_1$ ,  $\tau_0$ ,  $\tau_1$  and  $\sigma$ :*

```
print(chickresult, pars=c("b0", "b1", "tau0", "tau1", "sigma"))
```

```
## Inference for Stan model: chicks.
## 4 chains, each with iter=500; warmup=250; thin=1;
## post-warmup draws per chain=250, total post-warmup draws=1000.
##
##      mean se_mean   sd 2.5% 25% 50% 75% 97.5% n_eff Rhat
## b0    3.81      0 0.01 3.79 3.80 3.81 3.82  3.83   921 1.00
## b1    0.08      0 0.00 0.07 0.07 0.08 0.08  0.08   255 1.01
## tau0  0.06      0 0.01 0.03 0.05 0.06 0.07  0.09   128 1.02
## tau1  0.02      0 0.00 0.02 0.02 0.02 0.02  0.03   785 1.00
## sigma 0.11      0 0.00 0.10 0.10 0.11 0.11  0.11   666 1.00
##
## Samples were drawn using NUTS(diag_e) at Fri Apr 01 11:53:36 2022.
## For each parameter, n_eff is a crude measure of effective sample size,
```

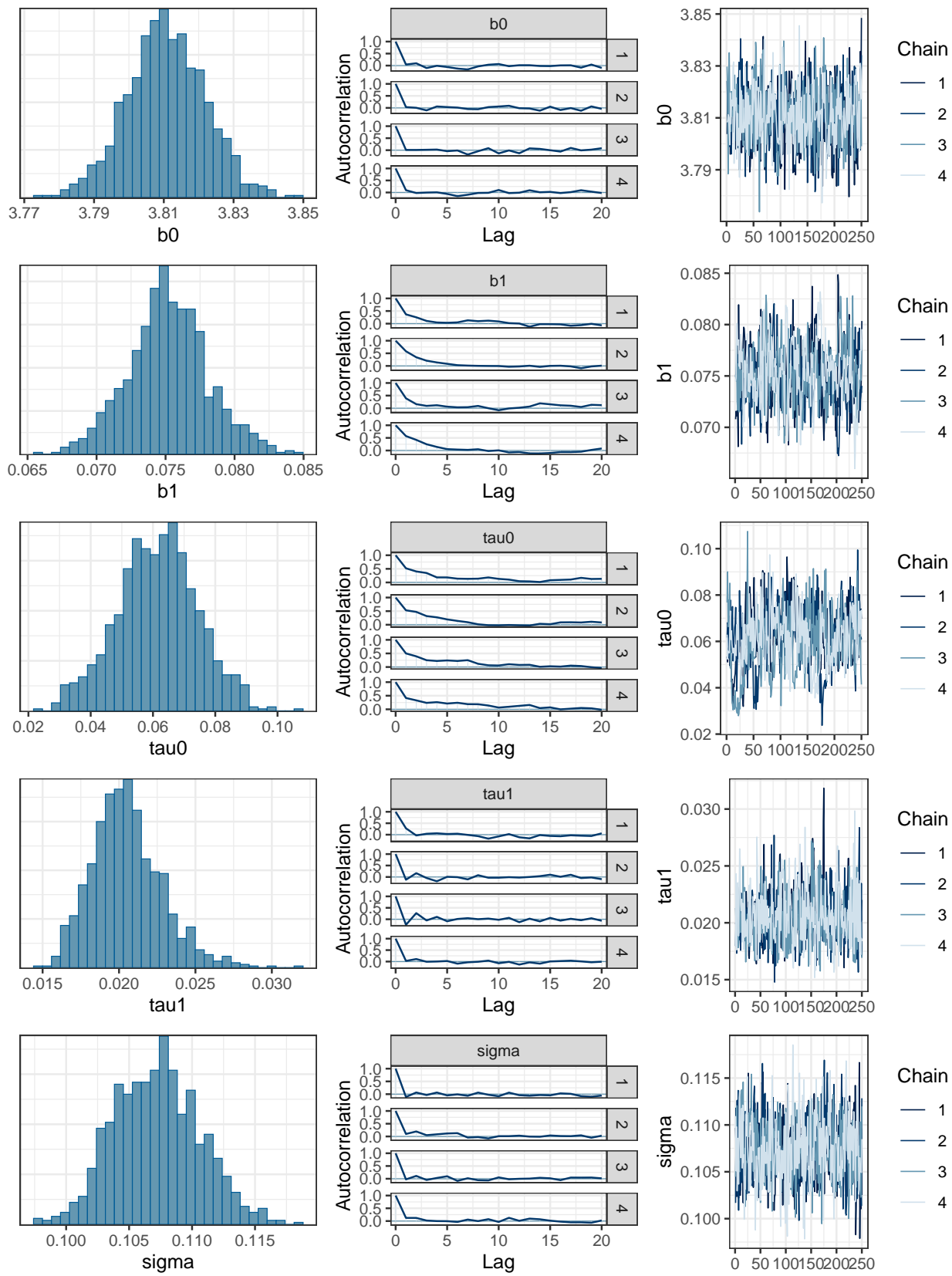
```
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

## Problem 1(b)

Use `shinystan` or `bayesplot` to generate and turn in the following plots, neatly organized:

- For  $\beta_0$ : (i) Histogram of the MCMC draws; (ii) Autocorrelation plot for at least one of the four Markov chains; (iii) trace plot of all four Markov chains on the same graph, with different colors to indicate the different Markov chains.
- For  $\beta_1$ : the same three things.
- For  $\tau_0$ : the same three things.
- For  $\tau_1$ : the same three things.
- For  $\sigma$ : the same three things.

```
param <- "b0"
g1 <- mcmc_hist(chickresult,pars=param)
g2 <- mcmc_acf(chickresult,pars=param)
g3 <- mcmc_trace(chickresult,pars=param)
param <- "b1"
g4 <- mcmc_hist(chickresult,pars=param)
g5 <- mcmc_acf(chickresult,pars=param)
g6 <- mcmc_trace(chickresult,pars=param)
param <- "tau0"
g7 <- mcmc_hist(chickresult,pars=param)
g8 <- mcmc_acf(chickresult,pars=param)
g9 <- mcmc_trace(chickresult,pars=param)
param <- "tau1"
g10 <- mcmc_hist(chickresult,pars=param)
g11 <- mcmc_acf(chickresult,pars=param)
g12 <- mcmc_trace(chickresult,pars=param)
param <- "sigma"
g13 <- mcmc_hist(chickresult,pars=param)
g14 <- mcmc_acf(chickresult,pars=param)
g15 <- mcmc_trace(chickresult,pars=param)
grid.arrange(g1,g2,g3,
              g4,g5,g6,
              g7,g8,g9,
              g10,g11,g12,
              g13,g14,g15,ncol=3)
```



## Problem 1(c)

Set `y <- log(ChickWeight$weight)`, and use `shinystan` or `bayesplot` to produce graphical posterior predictive checks (ppc's) for

- $T(y) = \text{mean}(y)$
- $T(y) = \text{sd}(y)$
- At least one other test statistic  $T(y)$  that produces an interesting result.

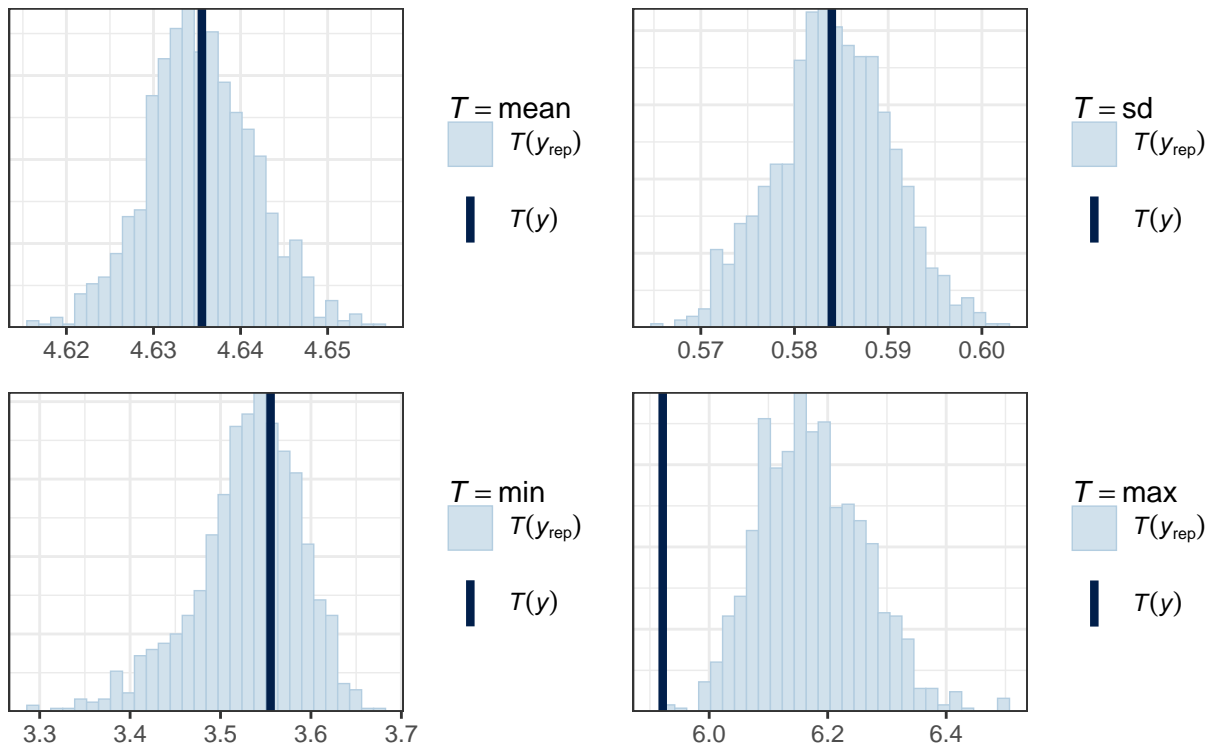
*In addition to the mean and the sd, I tried the min and the max as test statistics (see below). The mean, sd and min didn't show any misfit of the model to the data, but you can see that the max of the data is far outside the distribution of max's of data simulated from the model (from the posterior predictive distribution). That is interesting, because it suggests a feature of the data that the model isn't fitting very well.*

*Since the max of the real data is lower than the max of the simulated data, it suggests to me that perhaps  $\log(\text{weight})$  was too strong a transformation, and maybe something like  $\sqrt{\text{weight}}$  or some other fractional power would be better. (The fit would have to be much much better for the fractional power, though, since "log" is so much easier to explain to a client or colleague than a fractional power of weight would be.)*

```
y <- log(ChickWeight$weight)
yrep <- extract(chickresult, pars="yrep")$yrep

g1 <- ppc_stat(y, yrep, stat="mean")
g2 <- ppc_stat(y, yrep, stat="sd")
g3 <- ppc_stat(y, yrep, stat="min")
g4 <- ppc_stat(y, yrep, stat="max")

grid.arrange(g1, g2, g3, g4, ncol=2)
```



## Problem 1(d)

Now fit the corresponding `lmer()` model (the parameter estimates should be almost identical to your estimates from the `stan()` model). Turn in

- The plot of the conditional residuals for the `lmer()` model. If the model fits well, these should be approximately normally distributed.
- The plot of the "uniformized" conditional residuals<sup>4</sup> for the `lmer()` model, using `library(DHARMA)`. If the model fits, these should be approximately uniformly distributed.

Write a sentence or two identifying and commenting on any suggested improvements to the model, or any other useful information, that you can see in these plots.

```
lmer.1 <- lmer(log(weight) ~ Time + (Time|Chick), data=ChickWeight, REML=F)
```

*First I'll verify that the parameter estimates from `lmer()` and `stan()` are about the same...*

```
round(summary(lmer.1)$coef,2)
```

```
##           Estimate Std. Error t value
## (Intercept)    3.81         0.01  316.08
## Time           0.08         0.00   25.23
```

```
VarCorr(lmer.1)
```

```
## Groups   Name      Std.Dev. Corr
## Chick    (Intercept) 0.061207
##           Time       0.020369 -0.309
## Residual                    0.106655
```

```
print(chickresult, pars=c("b0", "b1", "tau0", "tau1", "sigma"))
```

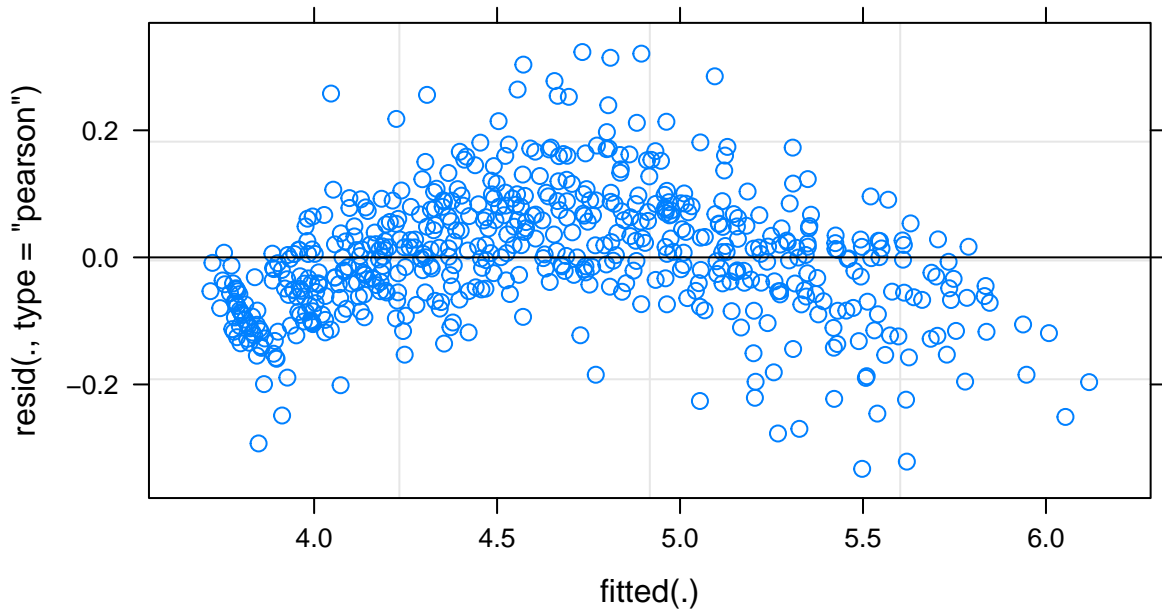
```
## Inference for Stan model: chicks.
## 4 chains, each with iter=500; warmup=250; thin=1;
## post-warmup draws per chain=250, total post-warmup draws=1000.
##
##      mean se_mean   sd 2.5% 25% 50% 75% 97.5% n_eff Rhat
## b0    3.81      0 0.01 3.79 3.80 3.81 3.82 3.83  921 1.00
## b1    0.08      0 0.00 0.07 0.07 0.08 0.08 0.08  255 1.01
## tau0  0.06      0 0.01 0.03 0.05 0.06 0.07 0.09  128 1.02
## tau1  0.02      0 0.00 0.02 0.02 0.02 0.02 0.03  785 1.00
## sigma 0.11      0 0.00 0.10 0.10 0.11 0.11 0.11  666 1.00
##
## Samples were drawn using NUTS(diag_e) at Fri Apr 01 11:53:36 2022.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

*...and then here are the plots...*

---

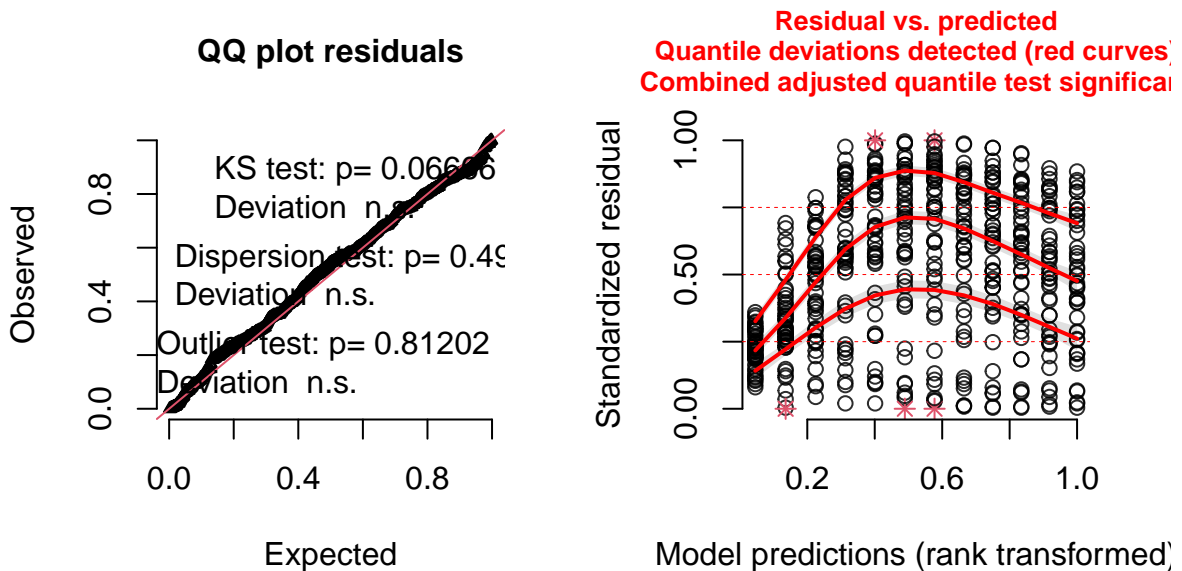
<sup>4</sup>We could build these by hand from the `stan()` model, but `DHARMA` is faster and less prone to user errors.

```
plot(lmer.1) ## "plot" plots conditional residuals for (g)lmer objects.
```



```
plot(simres <- simulateResiduals(lmer.1))
```

### DHARMA residual diagnostics



The qqplot looks good, though the KS test for uniformity of the "uniformized" residuals is just at the edge of significance. On the other hand, both residual vs fitted plots suggest that we missed some curvature in the model; I might try to fix this by adding  $I(\text{Time}^2)$  or  $I(\text{Time}^2) + I(\text{Time}^3)$  to the model.

## Problem 2 – Toenails

The `toenail` data frame comes from the `faraway` library, and you can access it by

```
install.packages("faraway")
library(faraway)
data(toenail)
```

There are 1908 observations from a study comparing two oral treatments for toenail infection. Patients were evaluated for the degree of separation of the nail. Patients were randomized into two treatments and were followed over seven visits—four in the first year and yearly thereafter. The patients have not been treated prior to the first visit so this should be regarded as the baseline. The variables (columns) are:

- ID: ID of patient
- outcome: 0=none or mild separation, 1=moderate or severe
- treatment: the treatment A=0 or B=1
- month: time of the visit (not exactly monthly intervals hence not round numbers)
- visit: the number of the visit

For most of this exercise we will be working with the model

$$\begin{aligned}\text{outcome}_i &\sim \text{Bernoulli}(p_i) \\ \text{logit}(p_i) &= \alpha_{0j[i]} + \alpha_{1j[i]} \cdot \text{month}_i + \beta_2 \cdot \text{treatment}_{j[i]} + \epsilon_i \\ \alpha_{0j} &= \beta_0 + \eta_{0j} \\ \alpha_{1j} &= \beta_1 + \eta_{1j} \\ \eta_{0j} &\overset{iid}{\sim} N(0, \tau_0^2) \\ \eta_{1j} &\overset{iid}{\sim} N(0, \tau_1^2)\end{aligned}$$

where  $i$  is the observation number and  $j$  is the patient ID number<sup>5</sup>. Note that there are some skips in the ID number sequence in the data set; it would be simplest for working with `stan()` to convert these ID numbers to successive integers with no skips. (Note that  $\sigma$  is missing from this model—why?)

(There is no  $\sigma$  in the model since the likelihood is Bernoulli, not normal, and for the Bernoulli, once you know the mean  $\mu_i = p_i$ , you also know the SD =  $\sqrt{p_i(1 - p_i)}$ .)

```
library(faraway)
```

```
data(toenail)
str(toenail)
```

```
## 'data.frame': 1908 obs. of 5 variables:
## $ ID : int 1 1 1 1 1 1 1 2 2 2 ...
## $ outcome : int 1 1 1 0 0 0 0 0 0 1 ...
## $ treatment: int 1 1 1 1 1 1 1 0 0 0 ...
## $ month : num 0 0.857 3.536 4.536 7.536 ...
## $ visit : int 1 2 3 4 5 6 7 1 2 3 ...
```

```
## You can see where the skips are in the ID numbers by doing this :
##
## diff(sort(unique(toenail$ID)))
##
## (if there were no skips you would see all 1's in the output of this command)
```

---

<sup>5</sup>Also note that in this model we are just letting  $\rho_{\eta_0, \eta_1} = 0$ , to make the `stan()` coding less complicated.

```
## ...and a very simple way to renumber the ID numbers so there are no skips is this:
ID.new <- factor(toenail$ID)
ID.new <- as.numeric(ID.new)

## You can verify that things line up correctly with
##
## View(cbind(toenail$ID,ID.new))
```

## Problem 2(a)

In the same folder as this assignment sheet there is a file `toenails.stan` with the *parameters* and *transformed parameters* sections left blank. Fill in the blank sections<sup>6</sup> and get the program to run using `library(rstan)`, taking at least 500 MCMC steps per chain<sup>7</sup>, and accepting the other `stan()` defaults, so that you end up with at least 1000 “good” MCMC samples from the four Markov chains produced. Turn in:

- The completed stan program `toenails.stan`.
- A printout of the fitted stan object but including only the lines for  $\beta_0$ ,  $\beta_1$ ,  $\beta_2$ ,  $\tau_0$ , and  $\tau_1$ .

Some notes:

- You are free to use `glmer()` to estimate the corresponding multi-level model, to make sure you are on the right track, but please don't turn in any `glmer()` code or output for this part of the exercise. Note that, even if you have done everything right the magnitudes of the `glmer()` estimates may not agree very well with the magnitudes of the `stan()` estimates; see part (d) below also.
- You will struggle to get a "great" run from `stan()` on this model. If you can get a run where most or all of the difficulties are in `yrep`, that will be good enough: the various convergence and stability diagnostics like `Rhat` don't really apply to `yrep` since it is simulated from the posterior predictive distribution; moreover the discreteness of `yrep` makes  $n_{eff}$  less meaningful.

*Clearly we have to define all of the parameters in the `model` block; the only tricky part is knowing what to put in the `parameters` block, vs. the `transformed parameters` block. But we can imitate what was done in `chicks.stan`, and what we come up with is this:*

```
print_file("toenails.stan")

## data {
##   int N;      // # observations
##   int J;      // # patients
##   int y[N];   // This should be the "outcome" variable
##              // in the "toenail" dataframe, i.e., 0's and 1's
##   real month[N];
##   int ID[N];  // convert ID in the dataframe to successive integers
##              // with no gaps
##   real treatment[N];
## }
##
## parameters{
##   real b0;
##   real b1;
##   real b2;
##   real eta0[J];
##   real eta1[J];
```

<sup>6</sup>You should not have to change any of the code already in the file; just add to it. However if it is easier for you to change some of the code in the file, that is fine too.

<sup>7</sup>If you need to take more steps to get convergence to the stationary distribution, that is fine.

```

##   real<lower=0> tau0;
##   real<lower=0> tau1;
## }
##
## transformed parameters {
##   real logit_p[N];
##   real a0[J];
##   real a1[J];
##   for (j in 1:J) {
##     a0[j] <- b0 + eta0[j];
##     a1[j] <- b1 + eta1[j];
##   }
##   for (i in 1:N) {
##     logit_p[i] <- a0[ID[i]] + a1[ID[i]]*month[i] + b2*treatment[ID[i]];
##   }
## }
##
## model {
##   for (i in 1:N) {
##     y[i] ~ bernoulli_logit(logit_p[i]) ;
##   }
##   for (j in 1:J) {
##     eta0[j] ~ normal(0,tau0);
##     eta1[j] ~ normal(0,tau1);
##   }
##   b0 ~ normal(0,1e6);
##   b1 ~ normal(0,1e6);
##   b2 ~ normal(0,1e6);
##   tau0 ~ uniform(0,50);
##   tau1 ~ uniform(0,50);
## }
##
## generated quantities {
##   real yrep[N];
##   for (i in 1:N) {
##     yrep[i] <- bernoulli_logit_rng(logit_p[i]);
##   }
## }

```

Now, consulting the *data* block to see what goes in the data list for *stan()*, we can use the following code to make a *stan* run:

```

attach(toenail)

ID.new <- factor(ID)
ID.new <- as.numeric(ID.new)

toedata <- list(N=length(ID.new),
               J=length(unique(ID.new)),
               y=outcome,
               month=month,
               ID=ID.new,

```

```

      treatment=treatment)
detach()

toemodel <- stan(file="toenails.stan",data=toedata,chains=0)

toeresult <- stan(fit=toemodel,data=toedata,iter=500)

...and the requested estimated parameters are:
print(toeresult,pars=c("b0","b1","b2","tau0","tau1"))

## Inference for Stan model: lineartoe.
## 4 chains, each with iter=500; warmup=250; thin=1;
## post-warmup draws per chain=250, total post-warmup draws=1000.
##
##      mean se_mean   sd  2.5%   25%   50%   75%  97.5% n_eff Rhat
## b0   -1.39     0.05  0.85 -3.14 -1.92 -1.43 -0.83  0.25   256 1.02
## b1   -1.56     0.04  0.24 -2.10 -1.71 -1.54 -1.39 -1.17    31 1.11
## b2   -2.20     0.06  1.17 -4.63 -2.90 -2.17 -1.38 -0.11   333 1.00
## tau0  8.37     0.18  1.10  6.60  7.55  8.25  9.06 10.69    37 1.09
## tau1  1.15     0.03  0.18  0.86  1.02  1.13  1.26  1.57    30 1.11
##
## Samples were drawn using NUTS(diag_e) at Fri Apr 01 14:15:53 2022.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).

```

The `stan()` result is not super (small `n_eff`'s and some `Rhat`'s bigger than 1.1). I could run the stan model longer (say with `iter=1000` or `iter=2000` rather than `iter=500`) to get better `Rhat`'s. but I'm going to go with what I already have from `stan()`; these `Rhat`'s are OK for just poking around with data and models as we are doing here.

(I would definitely try a higher number of iterations, and generally do what is needed to avoid warnings and things, if this were a final run for a formal paper or report!)

## Problem 2(b)

Use `shinystan` or `bayesplot` to generate and turn in the following plots, neatly organized:

- For  $\beta_0$ : (i) Histogram of the MCMC draws; (ii) Autocorrelation plot for at least one of the four Markov chains; (iii) trace plot of all four Markov chains on the same graph, with different colors to indicate the different Markov chains.
- For  $\beta_1$ : the same three things.
- For  $\beta_2$ : the same three things.
- For  $\tau_0$ : the same three things.
- For  $\tau_1$ : the same three things.

```

param <- "b0"
g1 <- mcmc_hist(toeresult,pars=param)
g2 <- mcmc_acf(toeresult,pars=param)
g3 <- mcmc_trace(toeresult,pars=param)
param <- "b1"
g4 <- mcmc_hist(toeresult,pars=param)
g5 <- mcmc_acf(toeresult,pars=param)
g6 <- mcmc_trace(toeresult,pars=param)

```

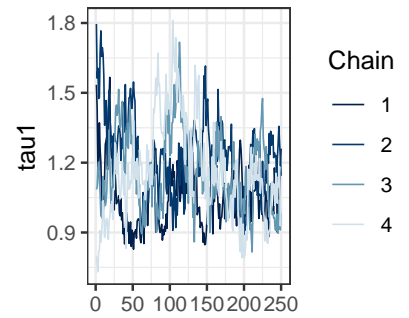
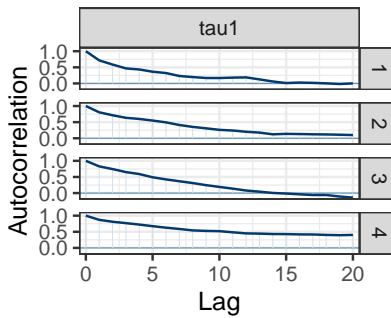
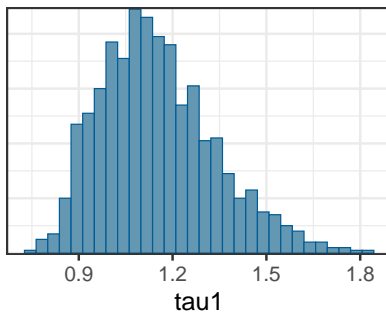
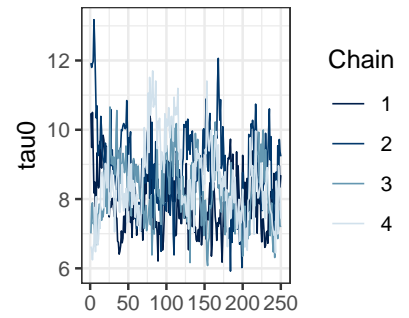
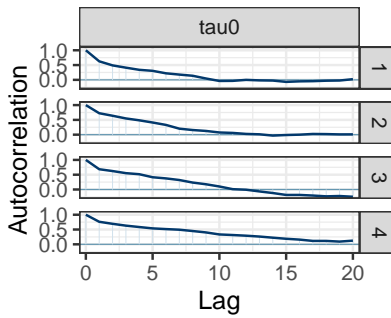
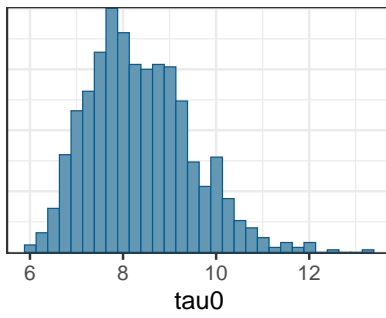
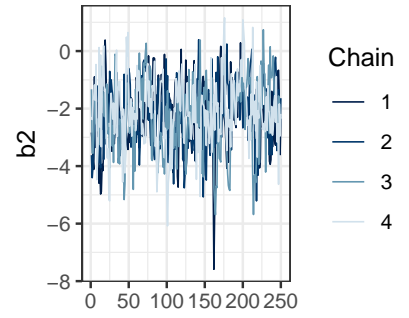
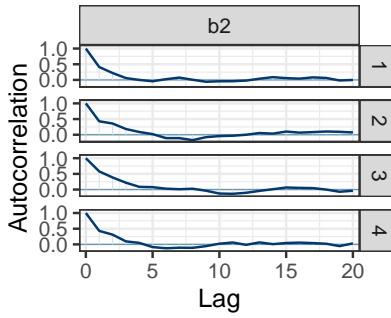
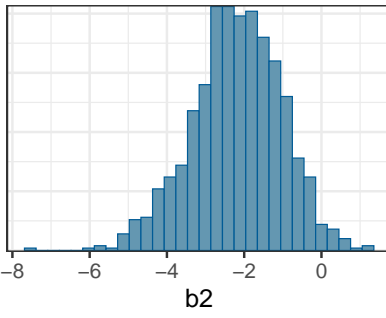
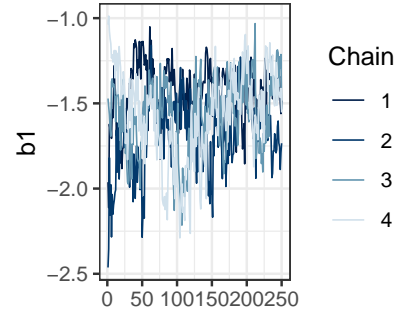
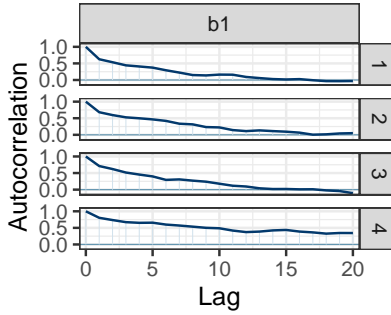
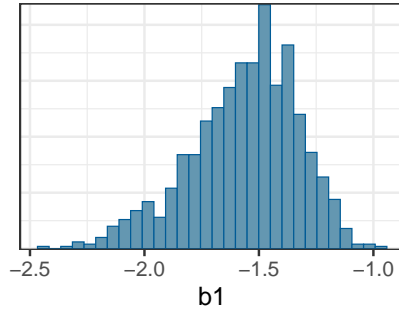
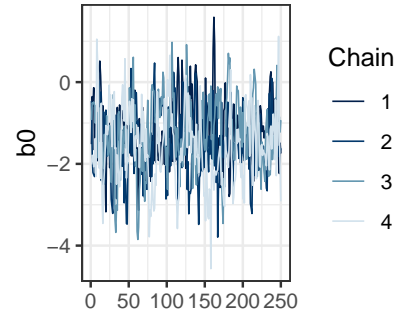
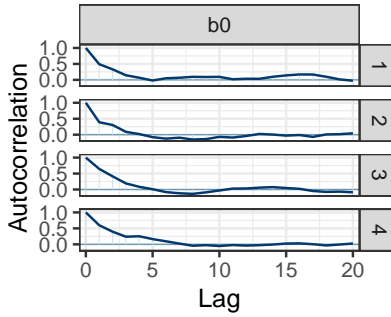
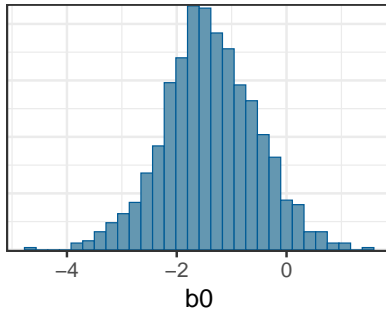
```

param <- "b2"
g7 <- mcmc_hist(toerresult,pars=param)
g8 <- mcmc_acf(toerresult,pars=param)
g9 <- mcmc_trace(toerresult,pars=param)
param <- "tau0"
g10 <- mcmc_hist(toerresult,pars=param)
g11 <- mcmc_acf(toerresult,pars=param)
g12 <- mcmc_trace(toerresult,pars=param)
param <- "tau1"
g13 <- mcmc_hist(toerresult,pars=param)
g14 <- mcmc_acf(toerresult,pars=param)
g15 <- mcmc_trace(toerresult,pars=param)

## The graphs are on the next page. You can see that the autocorrelations stay
## pretty large for b1, tau0 and tau1, which is consistent with the small n_eff's
## and large Rhat's we see for those parameters.

grid.arrange(g1,g2,g3,
              g4,g5,g6,
              g7,g8,g9,
              g10,g11,g12,
              g13,g14,g15,ncol=3)

```



## Problem 2(c)

Set `y <- toenail$outcome`, and use `shinystan` or `bayesplot` to produce graphical posterior predictive checks (ppc's) for

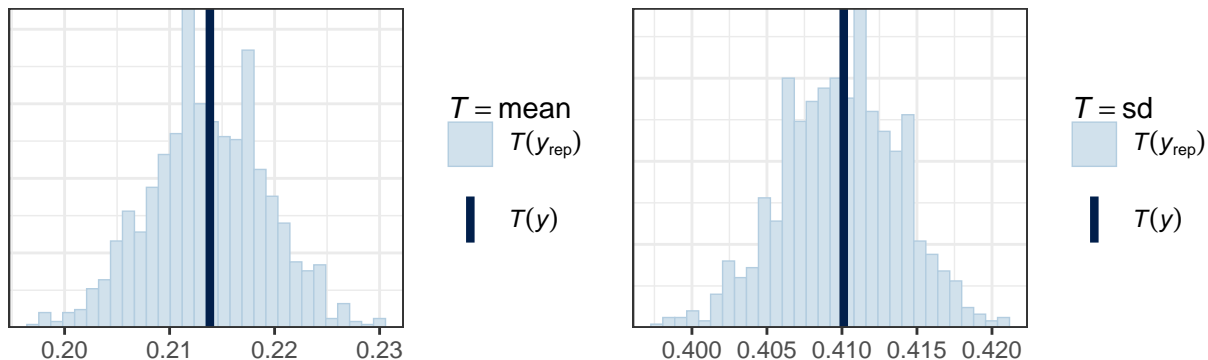
- $T(y) = \text{mean}(y)$
- $T(y) = \text{sd}(y)$
- At least one other test statistic  $T(y)$  that produces an interesting result.

Here are the plots for the mean and the SD. No indications of misfit here. I also tried the min and the max; nothing interesting there either.

```
y <- toenail$outcome
yrep <- extract(toeresult, pars="yrep")$yrep

g1 <- ppc_stat(y, yrep, stat="mean")
g2 <- ppc_stat(y, yrep, stat="sd")

grid.arrange(g1, g2, ncol=2)
```



As for coming up with one other test statistic, you may have had some creative idea that I didn't think of, but here's my idea:

After some thought, I decided to check to see if there were too many or too few positive residuals in different ranges of the fitted values  $\hat{p}_i$ , where  $p_i = P[\text{outcome}_i = 1]$ . The idea is this

- Divide the range of fitted values into three groups
  - The lower 1/3 of the range of fitted values  $\hat{p}_i$
  - The middle 1/3 of the range of fitted values  $\hat{p}_i$
  - The upper 1/3 of the range of fitted values  $\hat{p}_i$
- In each group:
  - Calculate  $T^{(obs)} = \text{number of positive residuals } (y_i - \hat{p}_i > 0) \text{ in the real data}$
  - For each replicated data set  $y^{rep,m}$ ,  $m = 1, \dots, M$ , calculate  $T^{(rep,m)} = \text{number of positive residuals } (y_i^{rep,m} - \hat{p}_i^{rep,m} > 0) \text{ in the } m^{th} \text{ replicated data set.}$
  - Compare  $T^{(obs)}$  to a histogram of  $T^{(rep,m)}$ 's,  $m = 1, \dots, M$ .

If for example there are too many positive residuals in the lower and upper thirds, but too few in the middle third, that suggests a U-shape for the residuals, and perhaps I would add  $I(\text{month}^2)$  to the model.

Here is the code to implement this idea, and the resulting plots:

```

##### Get the data elements needed

y <- toenail$outcome
## the real data

yrep <- extract(toeresult,pars="yrep")$yrep
## the replicated data sets; each row of this matrix is one replication of y

logit_p <- extract(toeresult,pars="logit_p")$logit_p
## the logits sampled from the posterior distribution again, each row of this
## matrix contains the logits for one replication yrep

p <- invlogit(logit_p)
## the p's used to generate the yrep's each row of this matrix contains the p's
## used to generate Bernoulli draws for one replication yrep

p_hat <- apply(p,2,mean)
## posterior mean estimate of p[outcome=1]

##### now, make the plots...

par(mfrow=c(3,1))

T.num.pos <- function(x) sum(x>0) ## the test statistic

##### Plot for the lower 1/3 of the range of p_hat

lo <- -1e6
hi <- min(p_hat) + (max(p_hat)-min(p_hat))/3
sel <- (p_hat<=hi) & (p_hat>lo)

T.obs <- T.num.pos((y-p_hat)[sel])
T.rep <- apply((yrep-p)[,sel],1,T.num.pos)

hist(T.rep,xlim=c(min(c(T.obs,T.rep)),max(c(T.obs,T.rep))),
     main="T=Number of Positive Residuals\n(Lower 1/3 of range of p_hat)",
     xlab="T(yrep)")
abline(v=T.obs,col="red",lwd=2)
text(x=T.obs,y=25,labels="T(y)",pos=4)

##### Plot for the middle 1/3 of the range of p_hat

lo <- min(p_hat) + (max(p_hat)-min(p_hat))/3
hi <- min(p_hat) + 2*(max(p_hat)-min(p_hat))/3
sel <- (p_hat<=hi) & (p_hat>lo)

T.obs <- T.num.pos((y-p_hat)[sel])
T.rep <- apply((yrep-p)[,sel],1,T.num.pos)

hist(T.rep,xlim=c(min(c(T.obs,T.rep)),max(c(T.obs,T.rep))),
     main="T=Number of Positive Residuals\n(Middle 1/3 of range of p_hat)",
     xlab="T(yrep)")
abline(v=T.obs,col="red",lwd=2)

```

```

text(x=T.obs,y=75,labels="T(y)",pos=4)

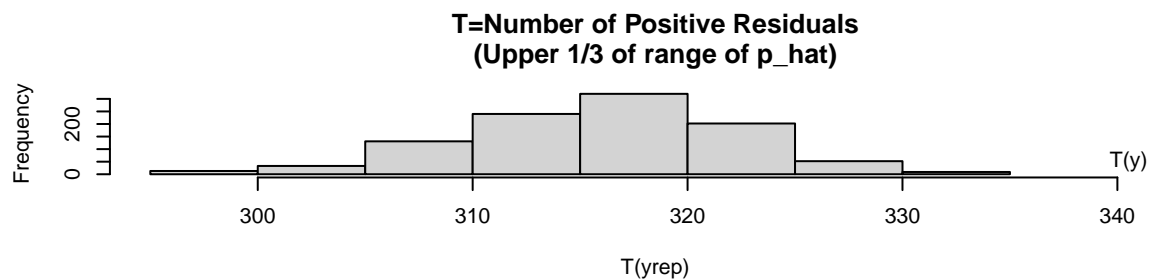
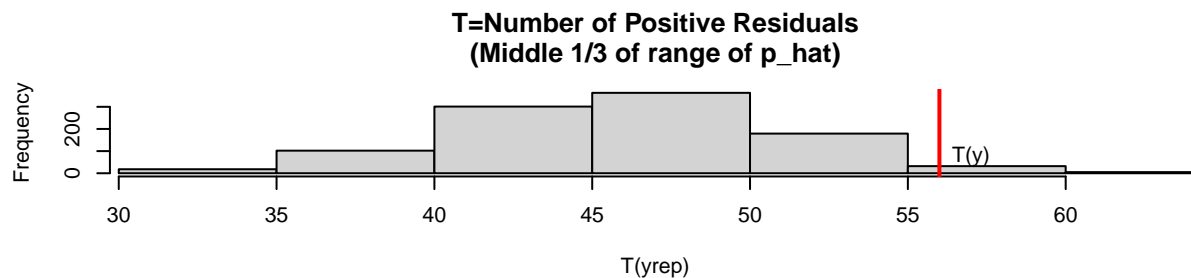
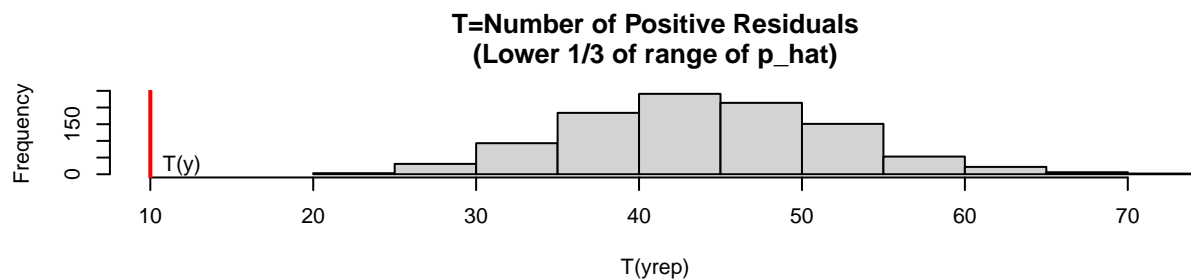
#### Plot for the upper 1/3 of the range of p_hat

lo <- min(p_hat) + 2*(max(p_hat)-min(p_hat))/3
hi <- +1e6
sel <- (p_hat<=hi) & (p_hat>lo)

T.obs <- T.num.pos((y-p_hat)[sel])
T.rep <- apply((yrep-p)[,sel],1,T.num.pos)

hist(T.rep,xlim=c(min(c(T.obs,T.rep)),max(c(T.obs,T.rep))),
     main="T=Number of Positive Residuals\n(Upper 1/3 of range of p_hat)",
     xlab="T(yrep)")
abline(v=T.obs,col="red",lwd=2)
text(x=T.obs,y=50,labels="T(y)",pos=2)

```



These plots show interesting misfit, but not the misfit I was expecting. Instead, there are too few positive residuals in the lower range, about the right number in the middle, and too many positive residuals in the upper range. That suggests that the residuals have an increasing trend, relative to the fitted model. Since the residual is  $y_i - \hat{p}_i$  and  $y_i$  can only be 0 or 1, this says that the model under-predicts the number of cases of  $y_i = 0$  when  $\hat{p}_i$  is low, and under-predicts the number of cases of  $y_i = 1$  when  $\hat{p}_i$  is high: the data is both zero-inflated and one-inflated, relative to the model predictions. There may be some other variable that we

could collect and include in the model, that might help the model to correctly predict these 0's and 1's.

## Problem 2(d)

Now fit the corresponding `glmer()` model (the parameter estimates should be almost identical to your estimates from the `stan()` model). When I did this, I thought that fixed effect estimates from `glmer()` were much less plausible than the fixed effect estimates from `stan()`, which made me trust the `stan()` results more. Do you agree? Why or why not?

```
glmer.1 <- glmer(outcome ~ month + treatment + (month||ID), data=toenail, family=binomial)
## note use of "||" in the random effect notation to force the correlation between eta's
## to be zero, just like the stan() model...
```

```
display(glmer.1)
```

```
## glmer(formula = outcome ~ month + treatment + (month || ID),
##       data = toenail, family = binomial)
##               coef.est coef.se
## (Intercept)  -12.31      1.22
## month        -12.50      1.27
## treatment    -0.03      1.54
##
## Error terms:
## Groups   Name      Std.Dev.
## ID       (Intercept) 72.74
## ID.1     month      11.87
## Residual                1.00
## ---
## number of obs: 1908, groups: ID, 294
## AIC = 1024.9, DIC = -766.6
## deviance = 124.1
```

The `glmer()` results just aren't very believable. The intercept is estimated to be  $\hat{\beta}_0 = -12.31$ ; the corresponding probability is  $1/(1 + \exp(-\hat{\beta}_0)) = 4.513131 \times 10^{-6}$  so that, without the random effects, the probability of a good outcome is essentially zero;  $\hat{\beta}_1 = -12.5$  is just as large, and the treatment effect  $\hat{\beta}_2 = -0.03$  is minuscule and very different from the estimate from `stan()`. The SD's of the random effects are huge however ( $\hat{\tau}_0 = 72.74$  and  $\hat{\tau}_1 = 11.87$ ), which suggests to me that the random effects, rather than the fixed effects, are picking up the interesting variation in  $p_i(\text{outcome} = 1)$  and in the treatment effect.

The estimates of the fixed effects from the `stan()` model ( $\hat{\beta}_0 = -1.39$ ,  $\hat{\beta}_1 = -1.56$  and  $\hat{\beta}_2 = -2.2$ ) and SD's of random effects ( $\hat{\tau}_0 = 8.37$  and  $\hat{\tau}_1 = 1.15$ ) are just much more believable.

## Problem 2(e)

Turn in

- The plot of the conditional residuals for the `glmer()` model. If the model fits well, these should be approximately normally distributed.
- The plot of the "uniformized" conditional residuals<sup>8</sup> for the `lmer()` model, using `library(DHARMa)`. If the model fits, these should be approximately uniformly distributed.

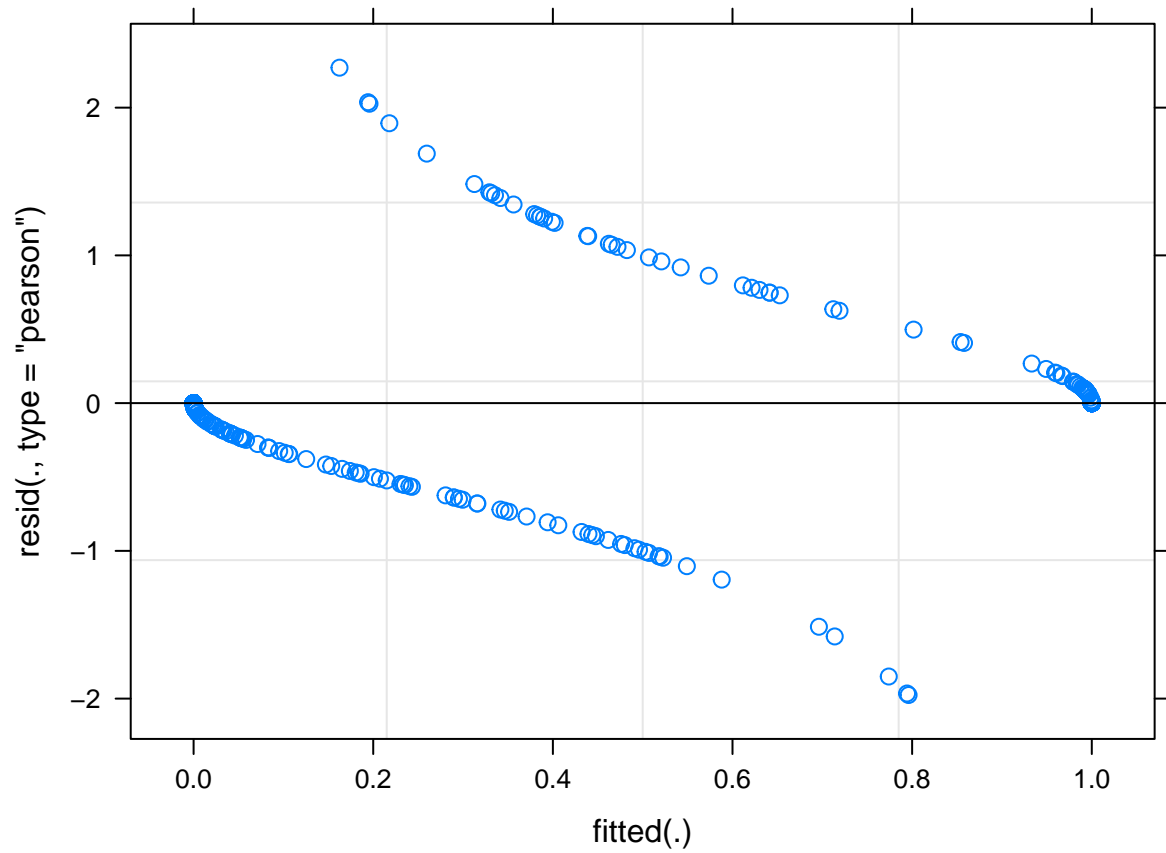
Write a sentence or two identifying and commenting on any suggested improvements to the model, or any other useful information, that you can see in these plots (or if you think some or all of the plots are useless,

---

<sup>8</sup>We could build these by hand from the `stan()` model, and we probably should since the `stan()` results seem more plausible than the `glmer()` results, but `DHARMa` is faster and less prone to user errors; it will be fine for the purpose of just getting used to using and interpreting `DHARMa` stuff.

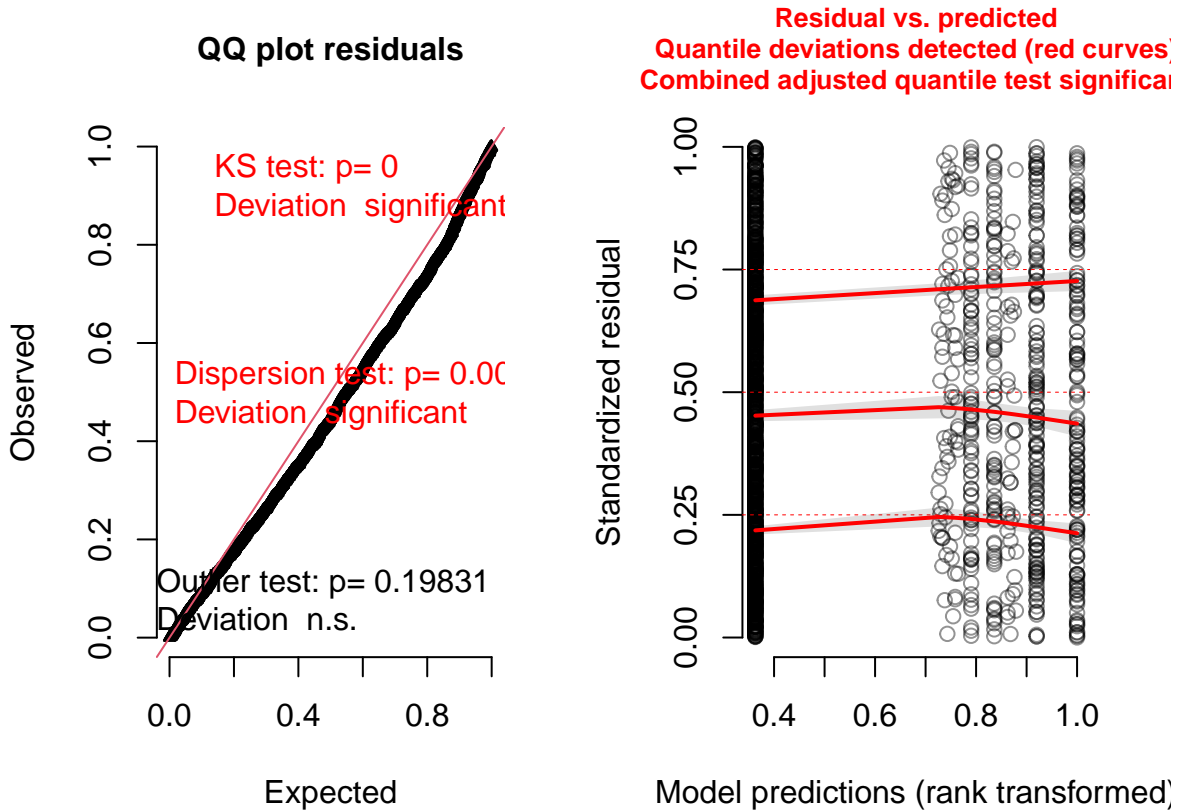
explain that as well).

```
plot(glmer.1)
```



```
plot(simres <- simulateResiduals(glmer.1))
```

## DHARMA residual diagnostics



The plot of the conditional residuals is virtually useless, just as we expect for untransformed residuals from a generalized linear model.

The plots of the uniformized residuals tell a somewhat different story.

In the qqplot on the left, there is some indication that may be a little long-tailed to the right (the plot curves upward from a straight line), the KS test for whether the “uniformized” residuals really follow the normal distribution is has a very low p-value (suggesting that the transformed data doesn’t really follow the normal distribution, as it would if the model fit well), and the dispersion test also has a low p-value, indicating that the residuals are over-dispersed (this over-dispersion could be related to the 0-inflation and 1-inflation we discovered with posterior predictive checks [ppc’s]).

In the residuals vs fitted plot on the right, the smooth curves that are supposed to be estimating the horizontal lines at the 25<sup>th</sup>, 50<sup>th</sup> and 75<sup>th</sup> percentiles, seem to be increasing (rather than horizontal) over most of their range. This too is consistent with what we discovered with ppc’s.