# 36-617: Applied Linear Models

Three Vignettes:

Prediction, Posterior Prediction, glm residuals

Brian Junker

132E Baker Hall

brian@stat.cmu.edu

# Announcements

- **Peer Reviews** *(Due Fri 1159pm)*
  - Reviews should be collegial and helpful.  Point out things the paper is doing right, and suggestions for improvement.
  - Write in the rubric categories provided, but do not assign points
- **Reading** *(in HW10 & weeks 13 & 14 folders on Canvas)*
  - Lynch, Ch 3 (read), Ch 4 (skim)
  - Lynch, Ch 9 (read)
- **HW10** *(Due Wed Dec 7, 1159pm)*
  - Just some “finger exercises” so you can play with estimating multilevel models with Stan, examining Stan output, etc.
- **Last Quiz** *(Mon-Tue Dec 5-6)*
  - Like midsemester survey – your thoughts about the class.

# Outline

- Vignette 1: Prediction
  - Review: Prediction with lm()
  - Prediction with glm()
  - Prediction with stan()
- Vignette 2: Posterior Prediction & Assessing Model Fit
  - Posterior Predictive Distribution
  - *If the model fits well, the real data should be indistinguishable from data simulated from the model*
- Vignette 3: Residuals for glm's
  - Fixing glm residuals using stan() [Bayes] or library(DHARMa) [glm/glmer]

# Prediction

- For ordinary linear regression, we may be interested in
  - Estimating $E[y_i|X_i]$
  - Predicting a new value $y^*$ at $X_i$
- Both are accomplished with the `predict()` function in R:
  - `predict(lm.1,interval="confidence")` gives a point estimate and CI for $E[y_i|X_i]$
    - the width for a 95% CI is $\approx 4 \cdot \widehat{SE}(E[y_i|X_i])$
  - `predict(lm.1,interval="prediction")` gives a point estimate and prediction interval for a new value $y^*$ at $X_i$
    - the width for a 95% PI is $\approx 4 \cdot \sqrt{\widehat{SE}(E[y_i|X_i])^2 + \hat{\sigma}^2}$

# Prediction with lm() – ChickWeight data…

```
attach(ChickWeight) ## comes with R...
lm.1 <- lm(log(weight) ~ Time*Diet + I(Time^2)*Diet)
predictdata <- data.frame(weight=NA,
  Time=c(12,12,30,30),
  Diet=factor(c(1,2,1,2),levels=c(1,2,3,4)))
pred.names <- c("  Time=12, Diet=1","  Time=12, Diet=2",
                "  Time=30, Diet=1","  Time=30, Diet=2")
pieces <- predict(lm.1,newdata=predictdata,se.fit=T)
CI.se <- pieces$se.fit
PI.se <- sqrt(pieces$se.fit^2 + pieces$residual.scale^2)
CI <- predict(lm.1,newdata=predictdata,
  interval="confidence")
row.names(CI) <- pred.names ; round(CI,2)
##                    fit  lwr  upr
##  Time=12, Diet=1 4.64 4.60 4.69
##  Time=12, Diet=2 4.81 4.75 4.87
##  Time=30, Diet=1 5.48 5.22 5.73
##  Time=30, Diet=2 5.48 5.15 5.82
CI <- cbind(fit=pieces$fit,
  lwr=pieces$fit-2*CI.se,upr=pieces$fit+2*CI.se)
row.names(CI) <- pred.names ; round(CI,2)
##                    fit  lwr  upr
##  Time=12, Diet=1 4.64 4.60 4.69
##  Time=12, Diet=2 4.81 4.75 4.87
##  Time=30, Diet=1 5.48 5.22 5.73
##  Time=30, Diet=2 5.48 5.14 5.82
PI <- predict(lm.1,newdata=predictdata,
  interval="prediction")
row.names(PI) <- pred.names ; round(PI,2)
##                    fit  lwr  upr
##  Time=12, Diet=1 4.64 4.22 5.06
##  Time=12, Diet=2 4.81 4.39 5.23
##  Time=30, Diet=1 5.48 4.99 5.96
##  Time=30, Diet=2 5.48 4.95 6.02
PI <- cbind(fit=pieces$fit,
  lwr=pieces$fit-2*PI.se,upr=pieces$fit+2*PI.se)
row.names(PI) <- pred.names ; round(PI,2)
##                    fit  lwr  upr
##  Time=12, Diet=1 4.64 4.22 5.07
##  Time=12, Diet=2 4.81 4.38 5.24
##  Time=30, Diet=1 5.48 4.98 5.97
##  Time=30, Diet=2 5.48 4.94 6.02
detach()
```

# How does this work for glm()?

- The `predict()` function "works" for `glm()` fits
- But the "`interval`" argument no longer does anything!
- The "`se.fit`" argument is still there
  - Based on experience with `lm()` predictions, `se.fit=T` probably returns $\widehat{SE}(E[y_i|X_i])$
  - We can use it for CI's for $E[y_i|X_i]$ but not for PI's for $y^*$ at $X_i$
- We can verify all this by fitting a stan() model with flat priors (so the results $\approx$ MLE results) and making predictions from the model

# Prediction with glm() – Roach data

```
roachdata <- read.csv("roachdata.csv",header=T)
glm.1 <- glm(y ~ roach1 + treatment + senior, offset =
  log(exposure2), data = roachdata, family=poisson)
predictdata <- data.frame(roach1=c(300,275,300,275),
   treatment=c(1,1,0,0), senior=rep(0,4),
   exposure2=rep(1,4), y=rep(NA,4))
pred.names <- c("  tx=1,roach1=300","  tx=1,roach1=275",
                "  tx=0,roach1=300","  tx=0,roach1=275")
pred.with.se <- predict(glm.1,
  newdata=predictdata,type="response", se.fit=T)
intervals.from.predict <- with(pred.with.se,
  data.frame(lower=fit-2*se.fit, fit=fit,
             upper=fit+2*se.fit))
row.names(intervals.from.predict) <- pred.names
round(intervals.from.predict,2)
##                    lower    fit  upper
##  tx=1,roach1=300 101.16 106.42 111.68
##  tx=1,roach1=275  85.19  89.37  93.55
##  tx=0,roach1=300 170.51 178.42 186.32
##  tx=0,roach1=275 143.60 149.84 156.08
```

- A good tip-off that these are CI's for $E[y_i|X_i]$, and not $y^*$ at $X_i$, is that they are rather narrow
  - The interval widths are ≈ 10-16 units
  - Dividing by 4, the SE's are ≈ 2.5-4
  - But the SE's should be something like sqrt(fit) for Poisson data:

```
round(sqrt(c(106.42,89.37,178.42,149.84)),2)
[1] 10.32  9.45 13.36 12.24
```

# Prediction with stan()

- We'll simulate "fake data" from a stan() model to explore this further
- This requires a new section in the stan() program, called "generated quantities"
- Each time new parameters are drawn from $f(params|data)$:
  - Dump them into vector beta
  - $\log(\lambda_{pred}) =$ predictdata · beta
  - Simulate y_pred ~ $Poiss(\lambda_{pred})$

```
model {
  real loglambda[N];
  for (i in 1:N) {
       loglambda[i] <- b0 + b_roach1*roach1[i] +
          b_treatment*treatment[i] +
          b_senior*senior[i] + offset[i];
       y[i] ~ poisson_log(loglambda[i]);
  }
  b0          ~ normal(0,1e6);
  b_roach1    ~ normal(0,1e6);
  b_treatment ~ normal(0,1e6);
  b_senior    ~ normal(0,1e6);
}
generated quantities {
  vector[5] beta;
  real y_pred[N_y_pred]; // number of rows in predictdata
  vector[N_y_pred] pred_loglambda;
  beta[1] <- b0;
  beta[2] <- b_roach1;
  beta[3] <- b_treatment;
  beta[4] <- b_senior;
  beta[5] <- 1; // for the offset
  pred_loglambda <- predictdata * beta; // mtx multiply!
  for (i in 1:N_y_pred) {
      y_pred[i] <- poisson_rng(exp(pred_loglambda[i]));
  }
}
```

The complete program is in stan_glm.1_text.stan

# Prediction with stan()

- We will focus on two matrices extracted from the stan() fit:

```
stan.Ey.df <- exp(as.data.frame(stan_glm.1_results)[,paste0("pred_loglambda[",1:4,"]")] )
dim(stan.Ey.df) ; names(stan.Ey.df) <- paste0("E[y_",1:4,"]")
## [1] 1000    4
head(stan.Ey.df,3) // 1000 draws from the posteriors of E[y] for the 4 predictions we want
##              E[y_1]           E[y_2]           E[y_3]           E[y_4]
## 1          109.1745         91.61842         175.9811         147.6820
## 2          104.4871         87.61899         180.0028         150.9437
## 3          105.6540         88.56638         181.4256         152.0833
## Etc.       Etc.             Etc.             Etc.             Etc.
stan.y.df <- as.data.frame(stan_glm.1_results)[,paste0("y_pred[",1:4,"]")]
dim(stan.y.df)
## [1] 1000    4
head(stan.y.df,3) // 1000 draws from the predictive distribution for the 4 predictions we want
##   y_pred[1] y_pred[2] y_pred[3] y_pred[4]
## 1        96        86       166       141
## 2       118        78       168       145
## 3       107        75       166       139
## Etc.    Etc.       Etc.     Etc.      Etc.
```

# Prediction with stan()

```
stan_glm.1_data <- c(as.list(roachdata),
  list(N=dim(roachdata)[1], N_y_pred=4,
  predictdata=cbind("(Int)"=1,
    roach1=c(300,275,300,275), treatment=c(1,1,0,0),
    senior=rep(0,4), logexposure2=log(rep(1,4)) ) ))
stan_glm.1_model <- stan(file="stan_glm.1_text.stan",
  data=stan_glm.1_data, chains=0)
init_stan_glm.1 <- function(){
  list(b0=rnorm(1), b_roach1=0.01*rnorm(1),
  b_treatment=rnorm(1), b_senior=rnorm(1))    }
stan_glm.1_results <- stan(fit=stan_glm.1_model,
  data=stan_glm.1_data, init=init_stan_glm.1,
  chains=4, iter=500)
stan.Ey.df <- exp(as.data.frame(stan_glm.1_results)[,
  paste0("pred_loglambda[",1:4,"]")] )
stan.Ey.with.se <- data.frame(est=apply(stan.Ey.df,2,mean),
  sd=apply(stan.Ey.df,2,sd) )
stan.Ey.intervals <- with(stan.Ey.with.se, data.frame(
  lower=est-2*sd, fit=est, upper=est+2*sd) )
row.names(stan.Ey.intervals) <- pred.names
stan.y.df <- as.data.frame(stan_glm.1_results)[,
  paste0("y_pred[",1:4,"]")]
```

```
stan.y.with.se <-
data.frame(est=apply(stan.y.df,2,mean),
  sd=apply(stan.y.df,2,sd))
stan.y.intervals <- with(stan.y.with.se, data.frame(
  lower=est-2*sd, fit=est, upper=est+2*sd))
row.names(stan.y.intervals) <- pred.names
round(intervals.from.predict,2)     //  from predict.glm
##                    lower    fit  upper
##  tx=1,roach1=300 101.16 106.42 111.68
##  tx=1,roach1=275  85.19  89.37  93.55
##  tx=0,roach1=300 170.51 178.42 186.32
##  tx=0,roach1=275 143.60 149.84 156.08
round(stan.Ey.intervals,2)          //  from stan()
##                    lower    fit  upper
##  tx=1,roach1=300 100.97 106.50 112.04
##  tx=1,roach1=275  85.05  89.45  93.85
##  tx=0,roach1=300 170.01 178.33 186.65
##  tx=0,roach1=275 143.21 149.77 156.33
round(stan.y.intervals,2)           //  from stan()
##                    lower    fit  upper
##  tx=1,roach1=300  84.79 106.61 128.44
##  tx=1,roach1=275  70.30  90.01 109.72
##  tx=0,roach1=300 150.84 178.40 205.97
##  tx=0,roach1=275 123.34 149.77 176.20
```

Additional details in the file prediction-in-(g)lmer and stan.r

# Posterior Prediction

- *Prediction is providing new values, $\boldsymbol{y_{pred}}$, based on the model and the data $\boldsymbol{y}$ that we've already seen.* The distribution of these new values is the *posterior predictive distribution*:

$$f(y_{pred}|y) = \int f(y_{pred}|\theta, y) f(\theta|y) d\theta = \int f(y_{pred}|\theta) f(\theta|y) d\theta$$

- We can simulate $y_{pred,m}$ as follows, for $m = 1, \ldots, M$:
  - Draw $\theta_{post,m} \sim f(\theta|y)$, the posterior distribution
  - Draw $y_{pred,m} \sim f(y_{pred,m}|\theta_{post,m})$, the likelihood evaluated at $\theta_{post,m}$.
- Then $y_{pred,1}, \ldots, y_{pred,M}$ will be draws from $f(y_{pred}|y)$
  - We can explore the quality of our predictions by exploring $y_{pred,1}, \ldots, y_{pred,M}$

# Posterior Predictive Distribution

- Generating a full replicate $y_{pred}$ of our original data $y$ is actually easier than generating the specific predictions in our previous example!
- Each time new parameters are drawn from $f(params|data)$:
  - Calculate $\log(\lambda)$ from them
  - Use $\log(\lambda)$ to specify the likelihood
  - Use $\log(\lambda)$ again to simulate y_pred $\sim Poiss(\lambda)$

```
transformed parameters {
  real loglambda[N];
  for (i in 1:N) {
       loglambda[i] <- b0 + b_roach1*roach1[i] +
           b_treatment*treatment[i] +
           b_senior*senior[i] + offset[i];
  }
}
model {
  for (i in 1:N) {
       y[i] ~ poisson_log(loglambda[i]);
  }
  b0           ~ normal(0,1e6);
  b_roach1     ~ normal(0,1e6);
  b_treatment  ~ normal(0,1e6);
  b_senior     ~ normal(0,1e6);
}
generated quantities {
  real y_pred[N];
  for (i in 1:N) {
      y_pred[i] <- poisson_rng(exp(loglambda[i]));
  }
}
```

The full stan() model is in stan_glm.1_ppd.stan

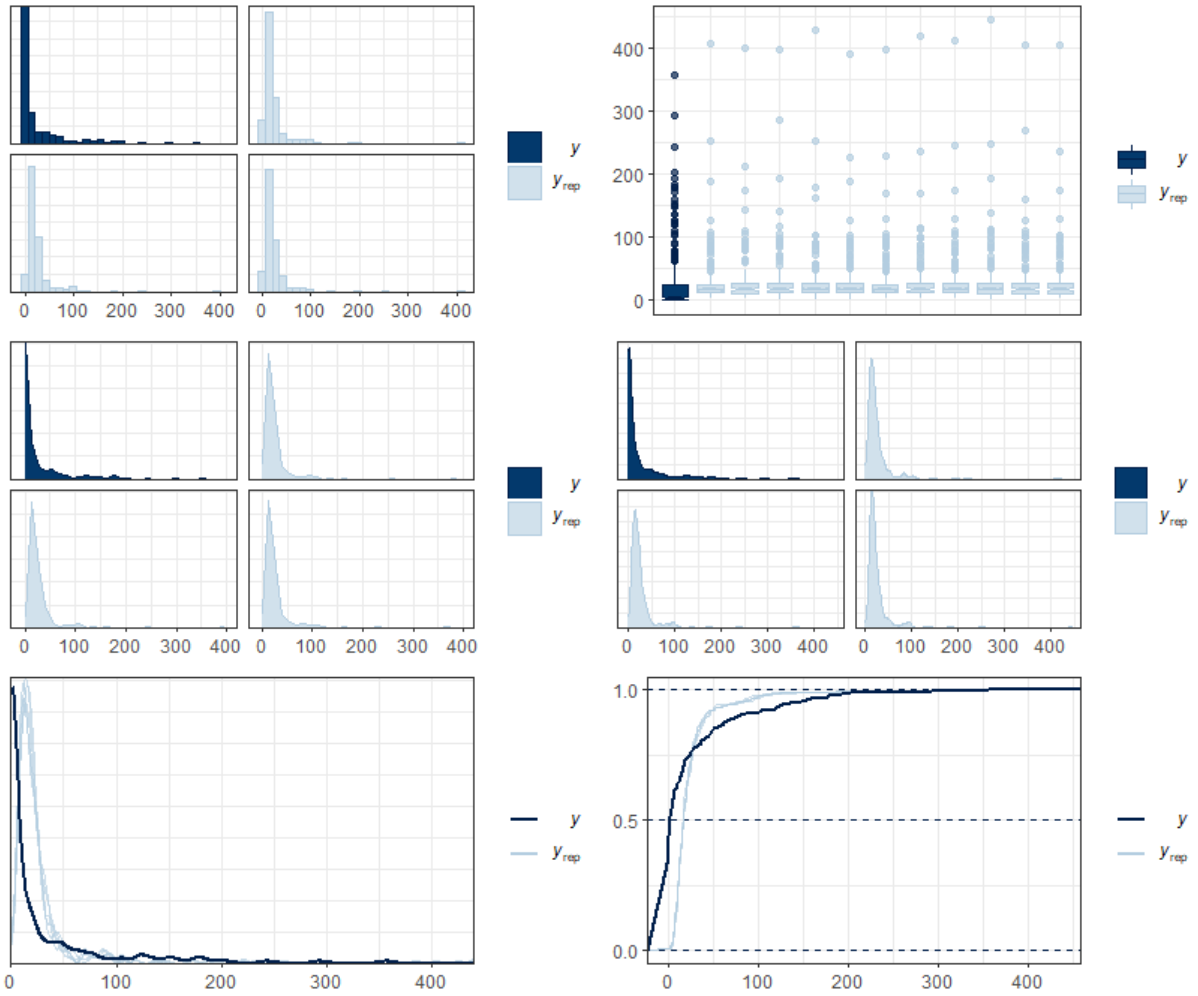
# Posterior predictive distribution – Roach data

- The posterior predictive distribution can be used to assess model fit
  - *"If the model fits well, the real data $\boldsymbol{y}$ should look like it belongs with data $\boldsymbol{y_{pred}}$ simulated from the model"*
- `shinystan` and `bayesplot` both can make graphical displays of simple "*posterior predictive checks*" of this proposition

```
stan_glm.1_ppd_data <- c(as.list(roachdata),
                         list(N=dim(roachdata)[1]))

stan_glm.1_ppd_model <- stan(file="stan_glm.1_ppd.stan",
                             data=stan_glm.1_ppd_data,
                             chains=0)

stan_glm.1_ppd_results <- stan(fit=stan_glm.1_ppd_model,
                               data=stan_glm.1_ppd_data,
                               init=init_stan_glm.1,
                               chains=4,
                               iter=1000)

## An earlier run with iter=500 had large rhats,
## suggesting it needed more iterations to converge to
## the stationary distribution.  Examination with
## shinystan confirmed this; so, increased to iter=1000.

y <- roachdata$y ## compare with y_pred in shinystan

launch_shinystan(stan_glm.1_ppd_results)
```

# Using bayesplot for ppc's

```
y <- roachdata$y
yrep <-
extract(stan_glm.1_ppd_results,pars="y_pred")$y_pred
Nrep <-  dim(yrep)[1] ; dim(yrep)
## [1] 2000  252
## yrep has 2000 simulated replications of the data y

g1 <- ppc_hist(y,yrep[sample(1:Nrep,3),])

g2 <- ppc_boxplot(y,yrep[sample(1:Nrep,11),])

g3 <- ppc_freqpoly(y,yrep[sample(1:Nrep,3),])

g4 <- ppc_dens(y,yrep[sample(1:Nrep,3),])

g5 <- ppc_dens_overlay(y,yrep[sample(1:Nrep,100),])

g6 <- ppc_ecdf_overlay(y,yrep[sample(1:Nrep,100),])

grid.arrange(g1,g2,g3,g4,g5,g6,ncol=2)
```



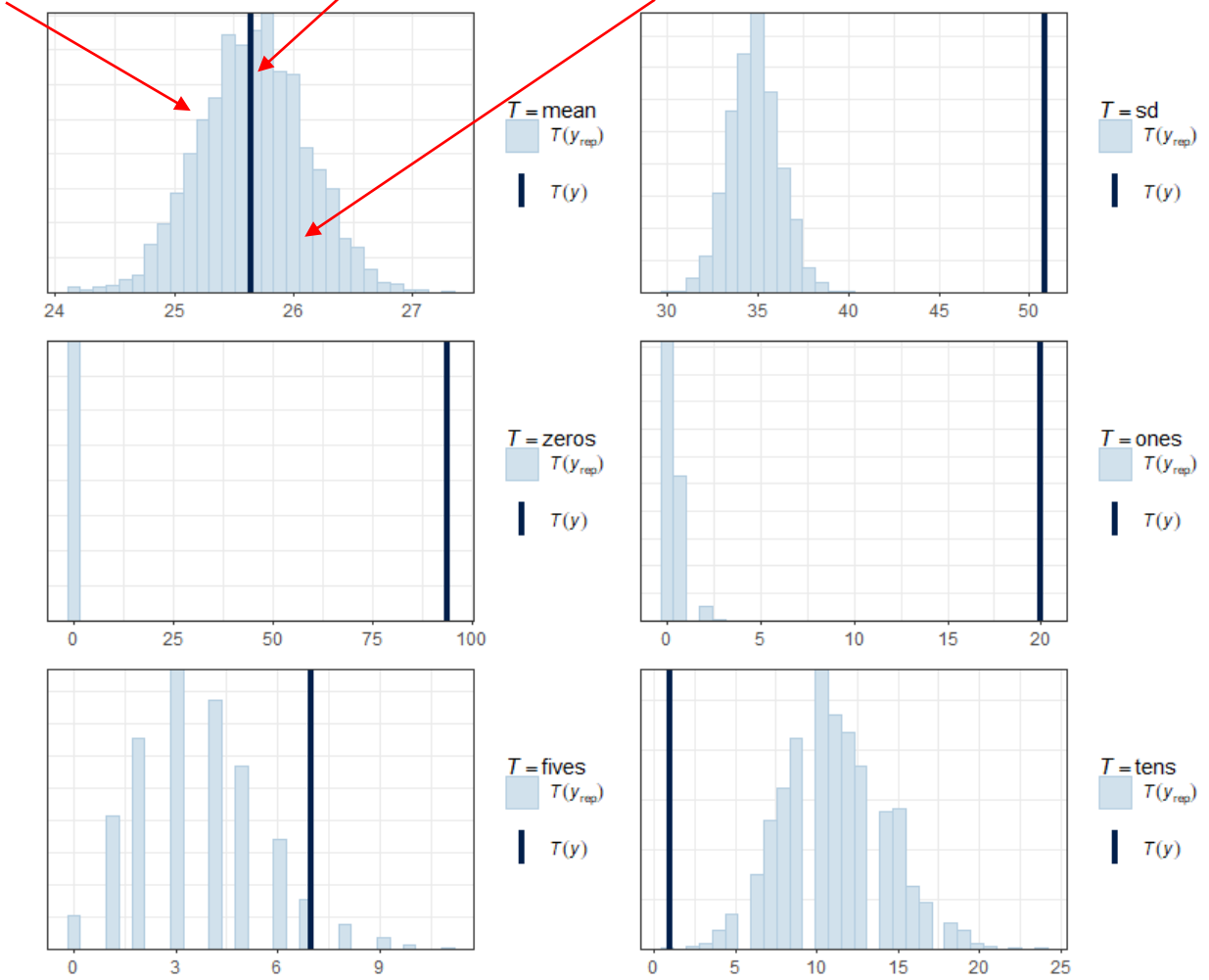
# Using bayesplot for ppc-based testing

```
g1 <- ppc_stat(y,yrep) ## default = mean

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

zeros <- function(y) { sum(y==0) }
g3 <- ppc_stat(y,yrep,stat=zeros)

ones <- function(y) { sum(y==1) }
g4 <- ppc_stat(y,yrep,stat=ones)

fives <- function(y) { sum(y==5) }
g5 <- ppc_stat(y,yrep,stat=fives)

tens <- function(y) { sum(y==10) }
g6 <- ppc_stat(y,yrep,stat=tens)

grid.arrange(g1,g2,g3,g4,g5,g6,ncol=2)
```

The simulated yrep's give the null distribution

The real data gives the test statistic

The tail probability is like a p-value



# Doing a ppc "by hand"

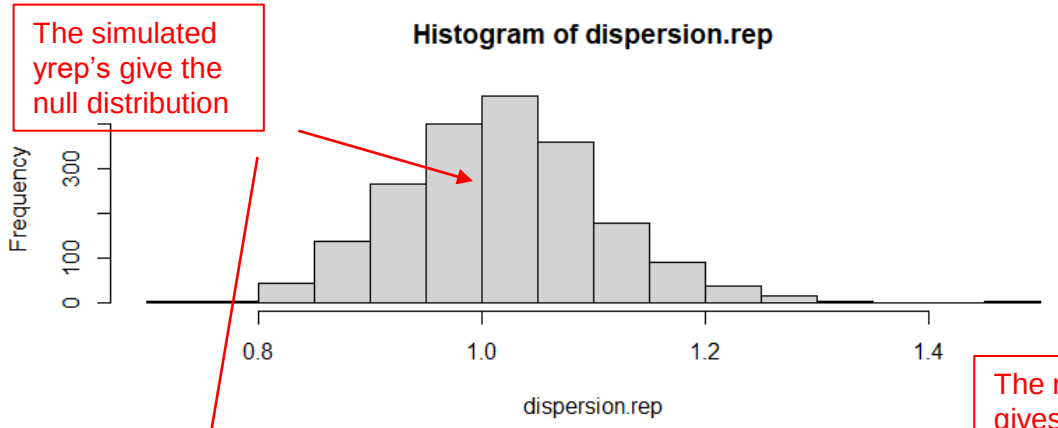
- Need to do "by hand" if your ppc also involves parameters…

```
lambdarep <-
exp(extract(stan_glm.1_ppd_results,pars="loglambda")
  $loglambda)
dim(lambdarep) ## [1] 200 262
lambdahat <- apply(lambdarep,2,mean)
dispersion <- function(y,lam,df) {
    sum((y-lam)^2/lam)/df  }
dispersion.obs <- dispersion(y,lambdahat,262-4)
dispersion.rep <- NULL
for (i in 1:Nrep) {
    dispersion.rep <- c(dispersion.rep,
          dispersion(yrep[i,],lambdarep[i,],262-4))  }
hist(dispersion.rep)
hist(dispersion.rep,xlim=c(0,dispersion.obs))
abline(v=dispersion.obs,col="red")
text(x=dispersion.obs,y=10,label="dispersion.obs",
  cex=0.75,adj=c(1,0.8))
```

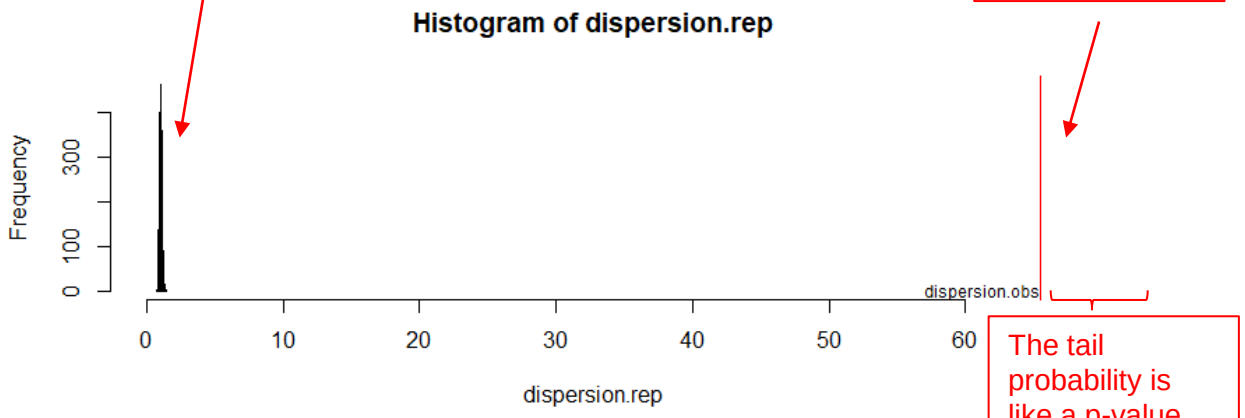
(test statistic)

(null distrib.)

The simulated yrep's give the null distribution




The real data gives the test statistic



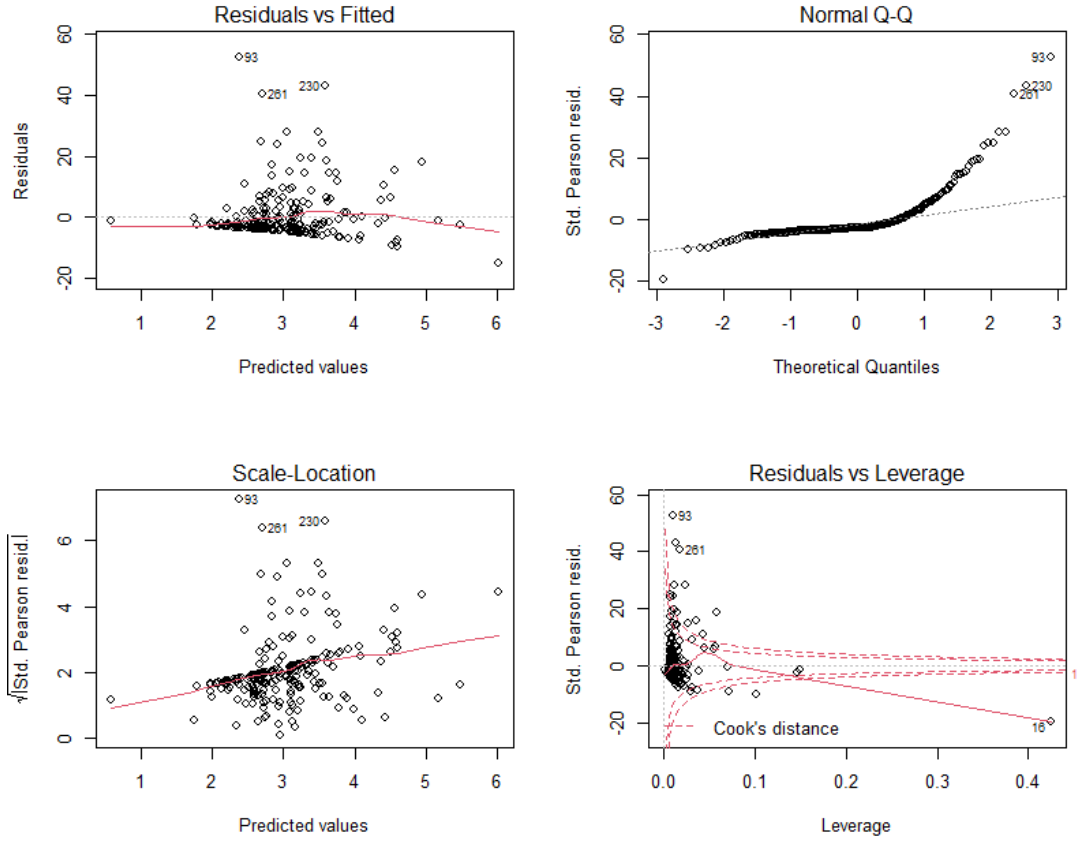

The tail probability is like a p-value

Ideally, $\sum \left(\frac{y_i-\lambda_i}{\sqrt{\lambda_i}}\right)^2 \sim \chi^2_{df}$ under $H_0$: no dispersion

# Fixing glm() residuals…

- glm() residuals generated by R are hard to interpret
- Problems include:
  - Discrete responses $y$ and range restrictions force curvilinear structure on residuals, even when the model fits
  - The asymptotics for the CLT for residuals don't work well when $y$ can only take on a few discrete values
  - Nonconstant variance is typical even when the model fits

```
par(mfrow=c(2,2))
plot(glm.1)
```



# A potential solution to glm() residuals…

- When the model fits, the residuals do have a well-defined distribution, even if it is not normal
- Use a posterior predictive simulation to get the (empirical) CDF $F_X(x)$ of that distribution
- Use the probability integral transform (PIT) $U = F_X(X)$ to try to change the distribution of the residuals to the uniform distribution
  - If the model fits well, then putting the real data residuals through the PIT based on replications from the model should make them look uniform
  - If the model doesn't fit the real data well, there should be interpretable patterns in the transformed residuals

PIT reminder: If $X \sim F_X(x) = P[X \leq x]$, then we know $U = F_X(X) \sim Unif(0,1)$

# Details about this potential solution…

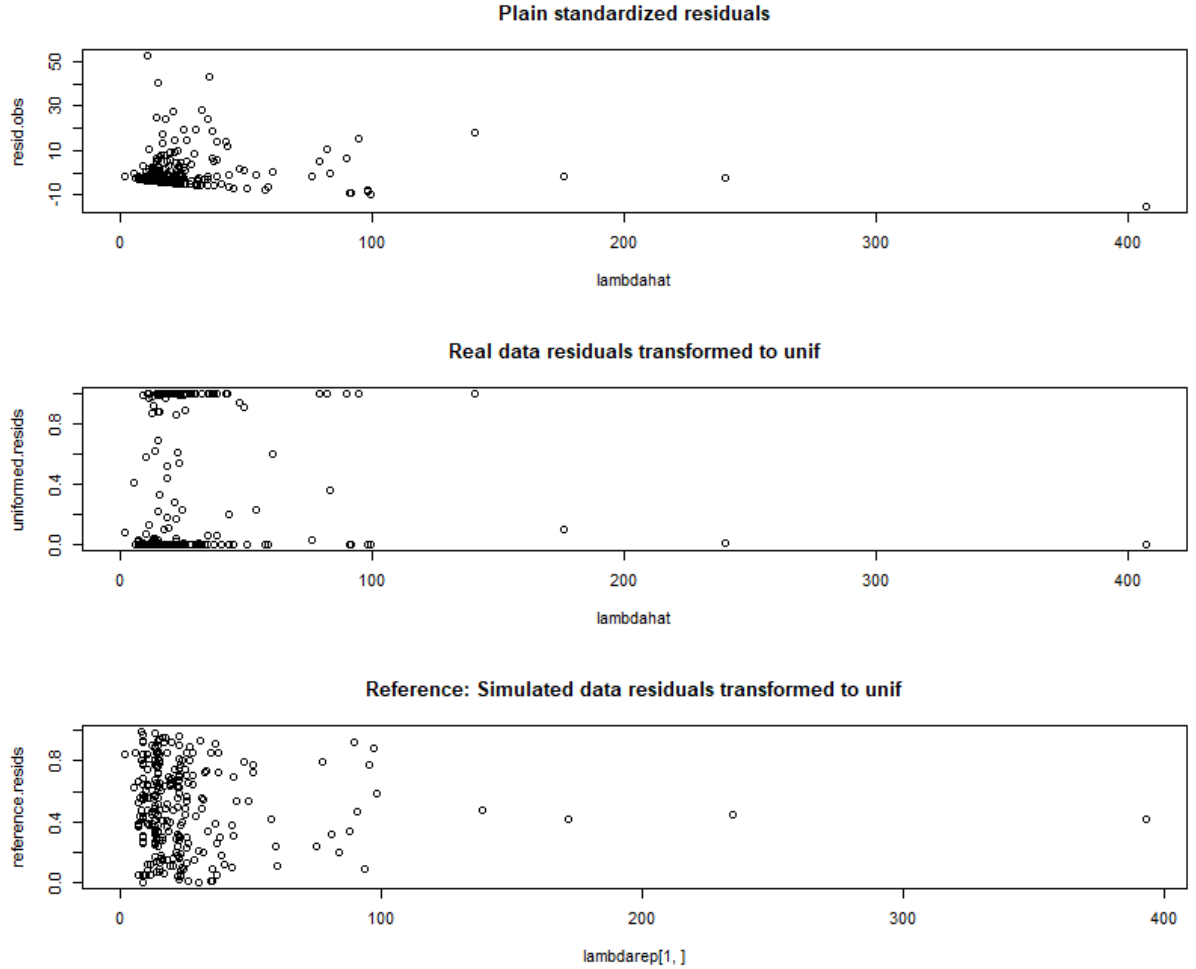
- Let $r_i = \left(y_i - \hat{E}[y_i]\right) / \widehat{SE}(\hat{E}[y_i])$ be standardized residuals of the real data
- Let $r_i^{rep,m} = \left(y_i^{rep,m} - \hat{E}[y_i]^{rep,m}\right) / \widehat{SE}(\hat{E}[y_i]^{rep,m})$ be the standardized residuals of the $m^{th}$ replication of simulated data

$$\begin{array}{ll} \text{real data} & \left\{ \begin{array}{cccc} r_1 & r_2 & \cdots & r_n \end{array} \right. \\ \text{simulated replications} & \left\{ \begin{array}{cccc} r_1^{rep,1} & r_2^{rep,1} & \cdots & r_n^{rep,1} \\ r_1^{rep,2} & r_2^{rep,2} & \cdots & r_n^{rep,2} \\ \vdots & \vdots & \ddots & \vdots \\ r_1^{rep,M} & r_2^{rep,M} & \cdots & r_n^{rep,M} \end{array} \right. \end{array}$$

- Use $r_1^{rep,1}, \ldots, r_1^{rep,M}$ to estimate $F_{r_1}(x)$,
  $r_2^{rep,1}, \ldots, r_2^{rep,M}$ to estimate $F_{r_2}(x)$, etc.
- If the model fits well, then $\hat{F}_{r_1}(r_1)$, $\hat{F}_{r_2}(r_2)$, …, $\hat{F}_{r_n}(r_n) \overset{approx}{\sim} Unif(0,1)$

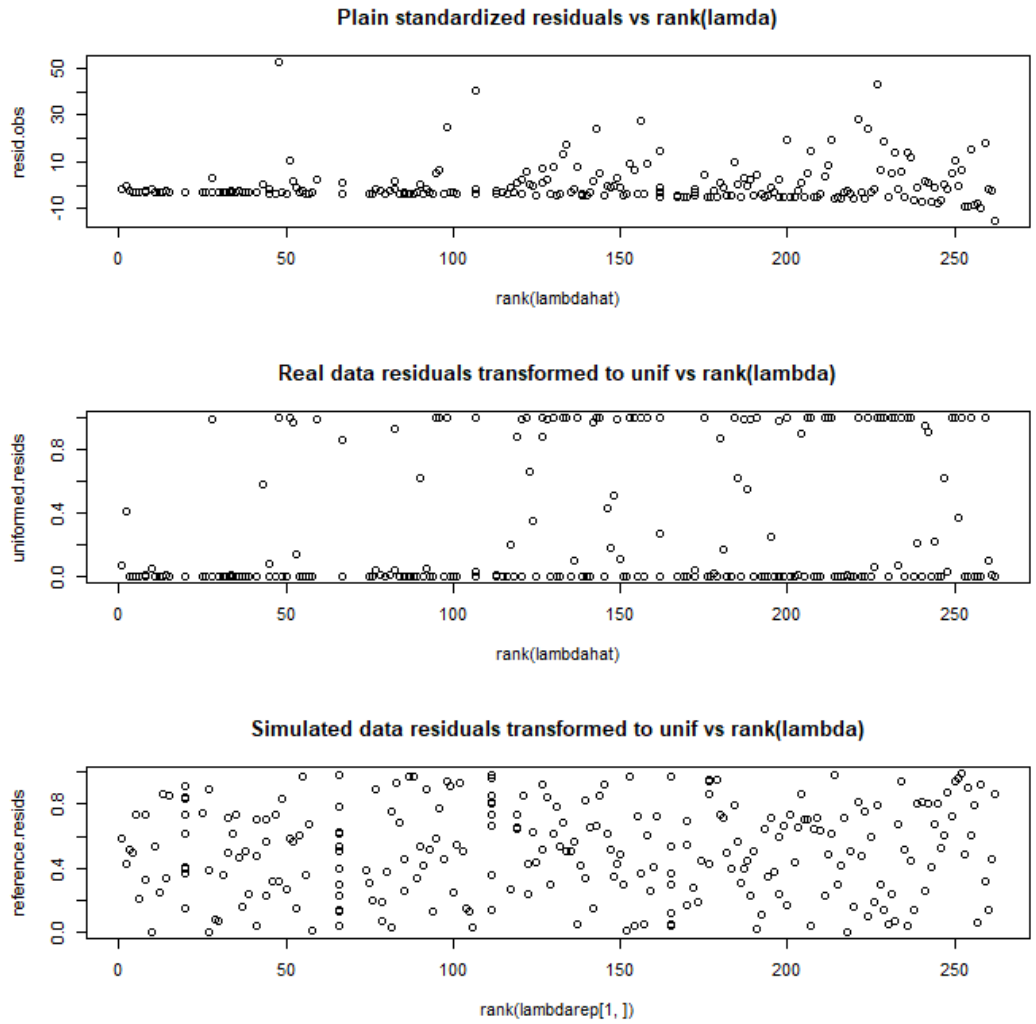
# The idea works, but it's not very legible…

```
ecdf.rep <- function(r, r.rep) {
    val <- rep(NA,length(r))
    for (i in 1:length(r)) {
        val[i] <- mean(r.rep[,i]<=r[i])
    }
    return(val)
}
resid.obs <- (y - lambdahat)/sqrt(lambdahat)
resid.rep <- (yrep - lambdarep)/sqrt(lambdarep)
uniformed.resids <- ecdf.rep(resid.obs,resid.rep)
reference.resids <- ecdf.rep(resid.rep[1,],
  resid.rep[-1,])
par(mfrow=c(3,1))
plot(resid.obs ~ lambdahat,
  main="Plain standardized residuals")
plot(uniformed.resids ~ lambdahat,
  main="Real data residuals transformed to unif")
plot(reference.resids ~ lambdarep[1,],
  main=paste("Reference: Simulated data residuals",
  "transformed to unif"))
```



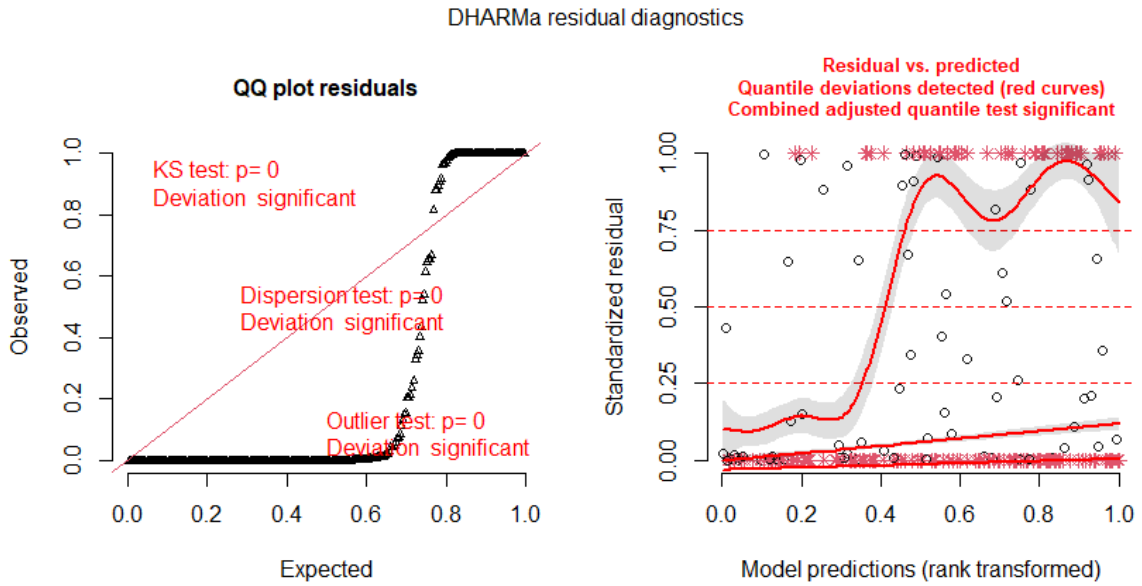
# For Legibility: Replace raw log($\lambda$) with its ranks

```
par(mfrow=c(3,1))

plot(resid.obs ~ rank(lambdahat),
  main="Plain standardized residuals")

plot(uniformed.resids ~ rank(lambdahat),
  main=
  "Real data residuals transformed to unif")

plot(reference.resids ~ rank(lambdarep[1,]),
  main=paste("Reference: Simulated data",
  "residuals transformed to unif"))
```

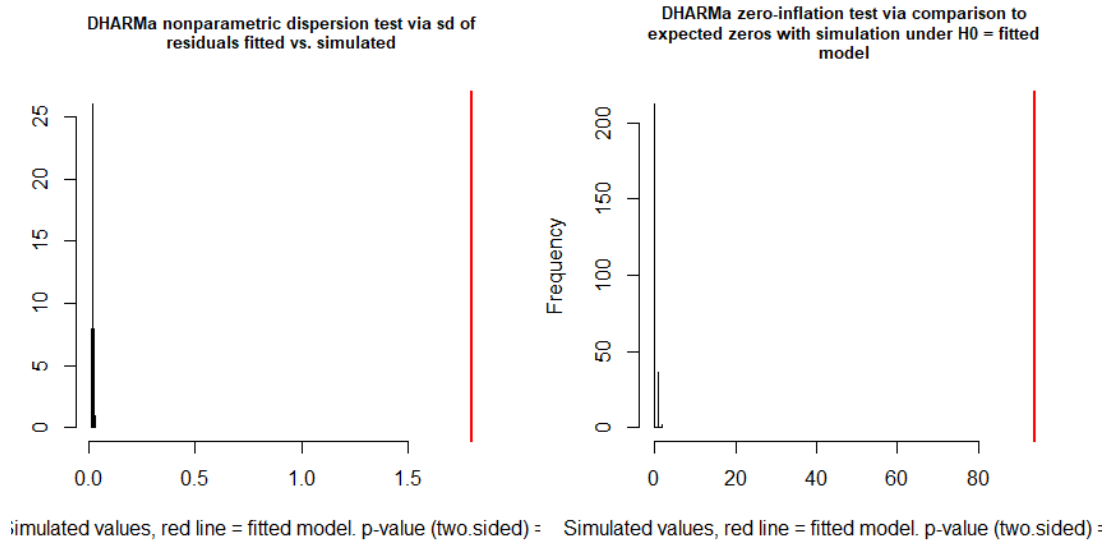


# For glm() and glmer(), it's easier to use library(DHARMa) rather than library(rstan)

```
library(DHARMa)
## vignette("DHARMa", package="DHARMa")
## is very helpful!

simulationOutput <- simulateResiduals(fittedModel =
  glm.1, plot = F)
plot(simulationOutput)
```



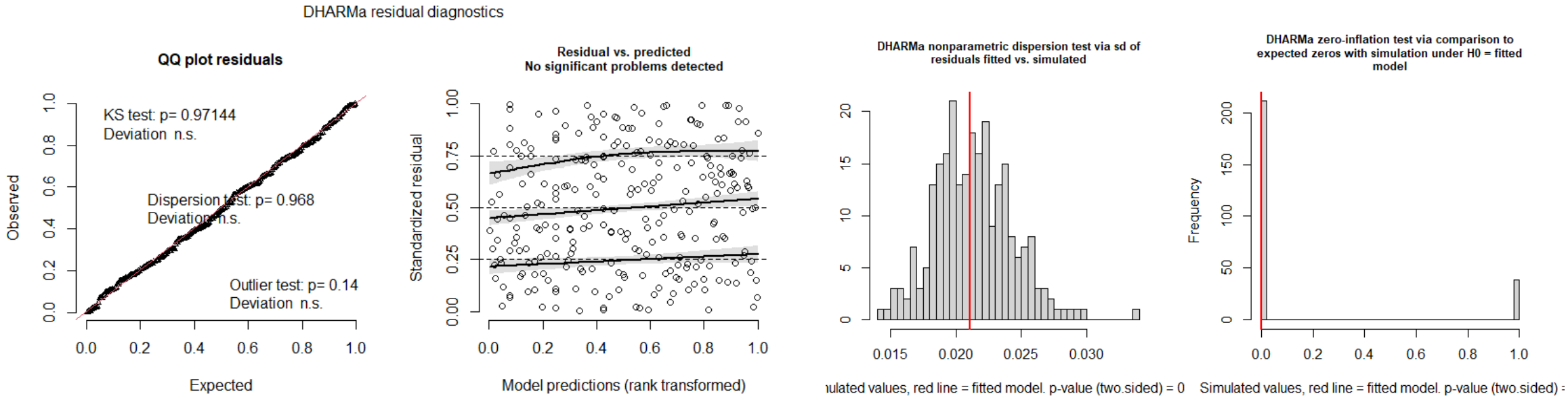

```
testDispersion(simulationOutput)

testZeroInflation(simulationOutput)
```



# Here is what the DHARMa plots look like when the model fits the data “perfectly”…

```
perfectmeans <- predict(glm.1,type="response")
y.perfect <- rpois(length(y),perfectmeans)
glm.1.perfect <- glm(y.perfect ~ roach1 + treatment +
  senior,offset=log(exposure2),data=roachdata,
  family=poisson)
sim.result <-
  simulateResiduals(fittedModel=glm.1.perfect,plot=T)
```

```
testDispersion(sim.result)

testZeroInflation(sim.result)
```



# Summary

- Vignette 1: Prediction
  - Review: Prediction with lm()
  - Prediction with glm()
  - Prediction with stan()
- Vignette 2: Posterior Prediction & Assessing Model FIt
  - Posterior Predictive Distribution
  - *If the model fits well, the real data should be indistinguishable from data simulated from the model*
- Vignette 3: Residuals for glm's
  - Fixing glm residuals using stan() [Bayes] or library(DHARMa) [glm/glmer]