```
1
2  # 2 日目
3  # 一般化線形モデル完全攻略
4
5  #  2013 年 8 月 28 最終改訂
6  #  執筆者 馬場真哉（北大水産 M2）
7  #  ウェブサイト http://logics-of-blue.com/
8  #  ミスなどのご連絡は logics.of.blue★gmail.com までお願いします。
9  #  （星を@に変更）
10
11 #==================================
12 # 予測と確率分布
13 # 実装する部分はありません。
14 # スライドに載せたグラフを描くプログラムだけです
15 #==================================
16
17 #  正規分布じゃダメな理由
18
19 lm.dist <- lm(dist ~ speed, data=cars)
20
21 new <- data.frame(
22   speed = seq(min(cars$speed), max(cars$speed), by = 0.1))
23 new
24 pred.lm <- predict(lm.dist, new, interval="prediction")
25 head(pred.lm)
26
27 plot(cars, ylim =c(-50,120), main="正規分布", cex.main=2, pch=16)
28 lines(new$speed, pred.lm[,1], lwd=3)
29 lines(new$speed, pred.lm[,2], lty = 2, lwd=2, col=4)
30 lines(new$speed, pred.lm[,3], lty = 2, lwd=2, col=4)
31 abline(h=0, lwd=4)
32
33
```

```
#============================
#  一般化線形モデル基礎
#============================

#-----------------------------
# スライドに載せた数値の計算方法
#-----------------------------

#  サンプルデータ
y <- c(7,9,8,11)
plot(y, pch=16, ylim=c(0,15), cex=2)


#  ポアソン分布の期待値が 5 だった時、各データが生じる確率
lambda <- 5
dpois(7, lambda=lambda)
dpois(9, lambda=lambda)
dpois(8, lambda=lambda)
dpois(11, lambda=lambda)

#  ポアソン分布の定義通り計算
kaijou <- function(x) gamma(x+1)
kaijou(3)

exp(-lambda) * lambda^7 / kaijou(7)
dpois(7, lambda=lambda)

#  ポアソン分布の期待値が 9 だった時、各データが生じる確率
lambda <- 9
dpois(7, lambda=lambda)
dpois(9, lambda=lambda)
dpois(8, lambda=lambda)
dpois(11, lambda=lambda)


#  期待値別、ポアソン分布のグラフ
```

```
x <- 0:20
par(mfrow=c(1,2))
# λ=5 のポアソン分布
lambda <- 5
plot(dpois(x, lambda=lambda)~x, type="b", ylab="確率", main="ダメそう
λ=5")
points(7,dpois(7, lambda=lambda), pch=16, cex=2, col=2)
points(9,dpois(9, lambda=lambda), pch=16, cex=2, col=2)
points(8,dpois(8, lambda=lambda), pch=16, cex=2, col=2)
points(11,dpois(11, lambda=lambda), pch=16, cex=2, col=2)

# λ=9 のポアソン分布
lambda <- 9
plot(dpois(x, lambda=lambda)~x, type="b", ylab="確率", main="よさそう
λ=9")
points(7,dpois(7, lambda=lambda), pch=16, cex=2, col=2)
points(9,dpois(9, lambda=lambda), pch=16, cex=2, col=2)
points(8,dpois(8, lambda=lambda), pch=16, cex=2, col=2)
points(11,dpois(11, lambda=lambda), pch=16, cex=2, col=2)
par(mfrow=c(1,1))


#  λ=5 の時の尤度
dpois(lambda=5, y)
prod(dpois(lambda=5, y))

#  対数尤度。総積に対してログをとる
log(prod(dpois(lambda=5, y)))

# 先に対数をとると、総和で計算できる
sum(log(dpois(lambda=5, y)))


#  λ=9 の時の尤度
dpois(lambda=9, y)
prod(dpois(lambda=9, y))
```

```
106
107 #  対数尤度。総積に対してログをとる
108 log(prod(dpois(lambda=9, y)))
109
110 # 先に対数をとると、総和で計算できる
111 sum(log(dpois(lambda=9, y)))
112
113
114
115 #-----------------------------------------
116 # R の optim 関数による最尤法の実装
117 #-----------------------------------------
118
119 # 最尤推定値をグラフで確認
120 x <- seq(0,15, by=0.01)
121 y <- c(7,9,8,11)
122 kekka <- numeric()
123
124 # λを変えることによる尤度の変化
125 for(i in 1:length(x)){
126   kekka[i] <- prod(dpois(lambda=x[i], y))
127 }
128
129 kekka
130 plot(kekka~x, type="l", xlab="λ", ylab="確率", lwd=4)
131
132 #  力技で、最も尤度が高くなるλを見つけ出す
133 max(kekka)
134 kekka.list <- data.frame(x=x,kekka=kekka)
135 kekka.list
136 subset(kekka.list, kekka.list$kekka==max(kekka))
137
138
139 #  optim を使って最尤法してみる
140 #  尤度にマイナスをかけていることに注意
141 #  optim は「最小化」するための関数。最大化したければマイナス 1 をかける
```

```
f.Lik <- function(para){
  f.Lik <- -prod(dpois(lambda=para, y))
  return(f.Lik)
}

#  小さいほうが尤度が高いことを意味している
f.Lik(9)
f.Lik(10)
f.Lik(15)

#  実はこの最適化はやや困難。method を変えると結果が変わる
optim(fn=f.Lik, par=c(8), method="Brent", lower=0, upper=10)
optim(fn=f.Lik, par=c(8), method="BFGS")
optim(fn=f.Lik, par=c(8))



#  対数をとってから最尤法
#  対数の方が小さい値（10 の-10 乗とか）に強いため、
# 対数をとってから計算するのがセオリー
f.logLik <- function(para){
  f.logLik <- -sum(log(dpois(lambda=para, y)))
  return(f.logLik)
}

f.logLik(9)
f.logLik(10)


optim(fn=f.logLik, par=c(8), method="Brent", lower=0, upper=10)
optim(fn=f.logLik, par=c(8), method="BFGS")
optim(fn=f.logLik, par=c(8))

#  実は最尤推定値は単なる期待値に等しい
mean(y)

```

```
178
179 # ★★★★★★★★★★★★★★★★★★★
180 # ★          写経はじめ          ★
181 # ★★★★★★★★★★★★★★★★★★★
182
183
184 #  サンプルデータ
185 y <- c(7,9,8,11)
186 plot(y, pch=16, ylim=c(0,15), cex=2)
187
188
189 #  ポアソン分布の期待値が 5 だった時、各データが生じる確率
190 lambda <- 5
191 dpois(7, lambda=lambda)
192 dpois(9, lambda=lambda)
193 dpois(8, lambda=lambda)
194 dpois(11, lambda=lambda)
195
196
197 #  対数をとってから最尤法
198 #  対数の方が小さい値（10 の-10 乗とか）に強いため、
199 # 対数をとってから計算するのがセオリー
200 f.logLik <- function(para){
201   f.logLik <- -sum(log(dpois(lambda=para, y)))
202   return(f.logLik)
203 }
204
205 f.logLik(9)
206 f.logLik(10)
207
208 # 最適化
209 optim(fn=f.logLik, par=c(8), method="BFGS")
210
211 #  実は最尤推定値は単なる期待値に等しい
212 mean(y)
213
```

```
214
215 # ♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪
216 # ♪          写経終わり              ♪
217 # ♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪
218
219
220 #=======================================
221 #  x によって y が変動することを加味したポアソン回帰
222 # スライド参照
223 #=======================================
224
225 # ★★★★★★★★★★★★★★★★★★★
226 # ★          写経はじめ              ★
227 # ★★★★★★★★★★★★★★★★★★★
228
229 x <- c(1, 2, 3, 4)
230 y <- c(5, 7, 10, 15)
231
232 plot(y ~ x, pch=16, cex=2)
233
234 #  尤度や対数尤度を計算する
235
236 #  切片１、傾きは 0.2 だと仮定したときの尤度は？
237 para <- c(1,0.2)
238
239 a <- para[1]
240 b <- para[2]
241
242 #  exp の中身が線形予測子
243 #  link 関数が log なので、その逆関数である exp をとった
244 lambda <- exp(a + b*x)
245 lambda
246
247 #  確率の計算はおなじみの  d○○を使う
248 #  今回はポアソン分布なので  dpois
249 Lik <- dpois(y, lambda=lambda)
```

```
Lik

#  対数尤度
logLik <- log(dpois(y, lambda=lambda))
logLik

#  尤度
prod(Lik)

#  対数尤度
#  普通の尤度よりも扱いやすそうな数値になっていることに注目
sum(logLik)
log(prod(Lik))

#  対数尤度で最適化するための関数
ML <- function(para){
  a <- para[1]
  b <- para[2]
  lambda <- exp(a + b*x)
  logLik <- -sum(log(dpois(y,lambda=lambda)))
  return(logLik)
}


#  対数尤度を最大化
result <- optim(fn=ML,par=c(1,0.2))
result

# 推定されたλ
result.par <- result$par
result.par

best.lambda <- exp(result.par[1] + result.par[2]*x)
best.lambda

plot(y ~ x, pch=16, cex=3)
```

```
lines(best.lambda ~ x, lwd=4, col=2)

# ♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪
# ♪          写経終わり            ♪
# ♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪


#  以下参考

# x=1 の時の Y の確率分布
# x=1 の時の推定されたλ
best.lambda[1]

y <- 0:20
dpois(y, best.lambda[1])

plot(dpois(y, best.lambda[1]) ~ y, type="b", lwd=3, pch=16, cex=2)

dpois(0, best.lambda[1])
dpois(1, best.lambda[1])
dpois(5, best.lambda[1])
dpois(10, best.lambda[1])

# x が１～４の時の、各々の Y の確率分布
par(mfrow=c(2,2))
plot(dpois(y, best.lambda[1]) ~ y, type="b", lwd=3,
  pch=16, cex=1.5, main="x=1 の時の確率分布",
  xlab="", ylab="", cex.main=2)
plot(dpois(y, best.lambda[2]) ~ y, type="b", lwd=3,
  pch=16, cex=1.5, main="x=2 の時の確率分布",
  xlab="", ylab="", cex.main=2)
plot(dpois(y, best.lambda[3]) ~ y, type="b", lwd=3,
  pch=16, cex=1.5, main="x=3 の時の確率分布",
  xlab="", ylab="", cex.main=2)
plot(dpois(y, best.lambda[4]) ~ y, type="b", lwd=3,
  pch=16, cex=1.5, main="x=4 の時の確率分布",
```

```
322     xlab="", ylab="", cex.main=2)
323   par(mfrow=c(1,1))
324
325
```

```
#==========================
# 残差の説明と deviance
#==========================
x <- c(1, 2, 3, 4)
y <- c(5, 7, 10, 15)
plot(y ~ x, pch=16, ylim=c(3,17), cex=2)

#  もし、データと完全に一致する予測値を出せていたとしたら？
# データが 5 だった時は期待値 5 のポアソン分布…
dpois(5, lambda=5)
dpois(7, lambda=7)
dpois(10, lambda=10)
dpois(15, lambda=15)

# 最尤法で推定された予測値から、今回のデータが得られたとしたら？
dpois(5, best.lambda[1])
dpois(7, best.lambda[2])
dpois(10, best.lambda[3])
dpois(15, best.lambda[4])


#  データを完全に予測できていた時の対数尤度
logLik.full <-
  log(dpois(5, lambda=5)) +
  log(dpois(7, lambda=7)) +
  log(dpois(10, lambda=10)) +
  log(dpois(15, lambda=15))

#  最尤法の結果得られた最大化対数尤度
logLik.ML <-
  log(dpois(5, lambda=best.lambda[1])) +
  log(dpois(7, lambda=best.lambda[2])) +
  log(dpois(10, lambda=best.lambda[3])) +
  log(dpois(15, lambda=best.lambda[4]))


```

```
#  完璧な予測の対数尤度と、制約（λ一定）の時の最大化対数尤度の比較
logLik.full
logLik.ML
logLik.full  -  logLik.ML

# residual deviance  とは「完璧予測の対数尤度」と
# 「最尤法で計算された最大化対数尤度」の差
#  雰囲気は残差平方和に似ている
# 2 倍してあるのは尤度比検定の都合。
logLik.full - logLik.ML
(logLik.full - logLik.ML) * 2

# R の関数で計算してみた結果
model.poisson <- glm(y ~ x, family="poisson")
summary(model.poisson)

#  glm 関数の結果と optim で推定した結果がだいたい等しいことを確認
coef(model.poisson)
result.par

#  deviance
#  カンペキな予測の対数尤度と、計算されたモデルの対数尤度の差の 2 倍
deviance(model.poisson)
(logLik.full - logLik(model.poisson))*2

# 残差
#  Type = respons  は、Type = respons の予測値と実測値との差
resid(model.poisson, type="respons")

#  Type = deviance が本命
resid(model.poisson, type="deviance")


#  deviance 残差の計算方法
#  平方根をとるという行為を除き、基本的には前回計算した deviance と同じ雰囲気であ
ることがわかるはず
```

```
best.lambda <- predict(model.poisson, type="response")
best.lambda

sqrt( 2  * (log(dpois(5, lambda=5)) -
  log(dpois(5, lambda=best.lambda[1]))) )
sqrt( 2  * (log(dpois(7, lambda=7)) -
  log(dpois(7, lambda=best.lambda[2]))) )
sqrt( 2  * (log(dpois(10, lambda=10)) -
  log(dpois(10, lambda=best.lambda[3]))) )
sqrt( 2  * (log(dpois(15, lambda=15)) -
  log(dpois(15, lambda=best.lambda[4]))) )

# 正負の符号の確認
sign(y  -  best.lambda)

# devance 残差は type の指定をしなくても、デフォルトに指定されている
resid(model.poisson)

# deviance と残差(Type=deviance)の比較
#  かなり普通の「残差」っぽく扱える
sum(resid(model.poisson)^2)
deviance(model.poisson)

#  summary の結果
#  Wald 検定
#  t 検定の GLM バージョンだと思うとわかりよい
#  GLM の場合は、各係数は正規分布に従うため、
#  t value ではなく,z value になっている
summary(model.poisson)
```

```
#=====================================================
# 尤度比検定
#  本当はサンプルサイズが大きい時しか使ってはいけない。
#  サンプルサイズ 4 で実行したのはあくまで勉強のため
#=====================================================

# ★★★★★★★★★★★★★★★★★★★
# ★          写経はじめ          ★
# ★★★★★★★★★★★★★★★★★★★


x <- c(1, 2, 3, 4)
y <- c(5, 7, 10, 15)
plot(y ~ x, pch=16, ylim=c(3,17), cex=2)

#  x によって y が変わると仮定したモデル
# 再掲
model.poisson <- glm(y  ~  x, family="poisson")
model.poisson
summary(model.poisson)

#  予測値(λ→平均値)がずっと一定だと仮定した NULL モデル
# ナイーブな予測を返す
#  NULL モデルの deviance が Null deviance
model.null <- glm(y  ~  1, family="poisson")
summary(model.null)

#  deviance の違い
deviance(model.null)
deviance(model.poisson)

#  最大化対数尤度の違い
logLik(model.poisson)
logLik(model.null)


```

```
#  尤度比検定！！
2 * (logLik(model.poisson) - logLik(model.null))
deviance(model.null) - deviance(model.poisson)

#  devianceの差→対数尤度の差の2倍→尤度の比　が統計量
#  この値が大きければxを入れたおかげでdevianceが大きく減ったことを意味する
#  ということは、この値が大きければ有意になりやすい
#  初日にやったF検定と「発想」はとてもよく似ている。
dev <- deviance(model.null) - deviance(model.poisson)
dev

#  p値
1 - pchisq(dev, df=1)

# Rの関数で確認
anova(model.poisson, test="Chisq")



# AIC
#  AIC  =  （最大化対数尤度  -  パラメタ数）  の2倍
AIC(model.poisson)
logLik(model.poisson)
(logLik(model.poisson) - 2) * -2
summary(model.poisson)



# ♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪
# ♪          写経終わり            ♪
# ♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪
```

```
#==================================
# もうちょっと本物っぽいデータを使って確認
#==================================

data4 <- read.csv("data4.csv")

plot(data4$y~data4$x)

#  尤度や対数尤度を計算する

#  切片１、傾きは 0.2 だと仮定したときの尤度は？
para=c(1,0.2)

a <- para[1]
b <- para[2]

#  exp の中身が線形予測子
#  link 関数が log なので、その逆関数である exp をとった
lambda <- exp(a + b*data4$x)

#  確率の計算はおなじみの  d○○を使う
#  今回はポアソン分布なので  dpois
Lik <- dpois(data4$y,lambda=lambda)
Lik

#  対数尤度
logLik <- log(dpois(data4$y,lambda=lambda))
logLik

#  尤度
prod(Lik)
#  対数尤度
#  普通の尤度よりも圧倒的に扱いやすそうな数値になっていることに注目
sum(logLik)


```

```
#  尤度で最適化するための関数
prod.ML <- function(para){
  a <- para[1]
  b <- para[2]
  lambda <- exp(a + b*data4$x)
  Lik <- -prod(dpois(data4$y,lambda=lambda))
  return(Lik)
}

#  対数尤度で最適化するための関数
ML <- function(para){
  a <- para[1]
  b <- para[2]
  lambda <- exp(a + b*data4$x)
  logLik <- -sum(log(dpois(data4$y,lambda=lambda)))
  return(logLik)
}


#  対数尤度を最大化
optim(fn=ML,par=c(1,0.2))
#  普通の尤度を最大化。うまくいってない。対数とるのは大事。
optim(fn=prod.ML,par=c(1,0.2))


#  R の glm 関数で計算
model.poisson <- glm(y~x, data=data4, family="poisson")
model.poisson
summary(model.poisson)

#  係数が optim の結果と一致していることを確認
coef(model.poisson)
optim(fn=ML,par=c(1,0.2))$par

#  最大化対数尤度の値も一致している
#  正負が違うのは、optim において、最小化するためマイナス 1 をかけているから
```

```
logLik(model.poisson)
optim(fn=ML,par=c(1,0.2))$value

#  予測してみる
new <- data.frame(x=seq(0,10,by=0.01))
new
pred <- predict(model.poisson, new, type="respons")

plot(data4$y~data4$x)
lines(pred~new$x, lwd=2)


# predict いろいろ
#  基本的には respons を使うことが多いと思う
#  しかしデフォルトはなぜか link。
predict(model.poisson, type="respons")
predict(model.poisson, type="link")
predict(model.poisson, type="term")
predict(model.poisson, type="term") + 2.7184 -
  predict(model.poisson, type="link")
exp(predict(model.poisson, type="link"))  -
  predict(model.poisson, type="respons")



# 予測区間
#  95%の確率で「データが」収まる区間
#  信頼区間（「引っ張られた線が」収まる区間）を出すのは省略

p.lower <- qpois(0.025, pred)
p.upper <- qpois(0.975, pred)

plot(data4$y~data4$x, xlim=c(0,10),ylim=c(-1,55),
   xlab="x", ylab="y", main="予測区間", cex.main=2)
lines(pred~new$x, lwd=2)
lines(p.lower~new$x, lwd=2, col=4)
```

```
lines(p.upper~new$x, lwd=2, col=2)
abline(h=0)


# ★★★★★★★★★★★★★★★★★★★
# ★          写経はじめ            ★
# ★★★★★★★★★★★★★★★★★★★


#  おまけ
#  r q p d の使い分け

#  r  は乱数

#  q  は値  →  確率がわかると値が決まる。

#  p  は累積確率       →値がわかると確率がわかる
#  d  はその時の確率  →値がわかると確率がわかる
#  ⇒p,dは確率、q,rは値


x <- seq(-2.5,2.5, by=0.01)
plot(dnorm(x)~x, type="l")
abline(v=qnorm(0.975))
abline(v=qnorm(0.025))

plot(pnorm(x)~x, type="l")
pnorm(qnorm(0.975))


# ♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪
# ♪          写経終わり            ♪
# ♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪


```

```
#==========================
# ポアソン回帰
# CPUE の標準化
#==========================
#  データの読み込み
data5 <- read.csv("data5_CPUE.csv")
summary(data5)
pairs(data5)

#  CPUE の計算
CPUE <- data5$Catch / data5$Effort
CPUE
data5 <- cbind(data5, CPUE)
head(data5)

# CPUE との関係性
par(mfrow=c(2,2))
plot(data5$Catch ~ data5$SST, main="漁獲量と海面水温の関係")
plot(data5$CPUE ~ data5$SST, main="CPUE と海面水温の関係")
plot(log(data5$CPUE)~data5$SST, main="CPUE にログをとった")
boxplot(data5$CPUE~data5$Year, main="CPUE と年の関係")
par(mfrow=c(1,1))


#  CPUE の年変動
Mean.CPUE <- as.numeric( tapply(data5$CPUE, data5$Year, mean) )
Mean.CPUE

Y <- 2001:2010
Y
plot(Mean.CPUE ~ Y, type ="l", xlim = c(2001,2010), lwd=2)
```

```
# ★★★★★★★★★★★★★★★★★★★
# ★          写経はじめ          ★
# ★★★★★★★★★★★★★★★★★★★

#  データの読み込み
data5 <- read.csv("data5_CPUE.csv")
summary(data5)
pairs(data5)

#  CPUE の計算
CPUE <- data5$Catch / data5$Effort
CPUE
data5 <- cbind(data5, CPUE)

#  CPUE のカウントモデル
#  努力量が倍になれば、漁獲量は倍になるはずなので、オフセット項を入れる。
model.CPUE <- glm(
  Catch ~ as.factor(Year) + offset(log(Effort)) + SST,
  family="poisson",
  data=data5)
summary(model.CPUE)
library(car)
Anova(model.CPUE)

# モデルの診断
#  Cook 距離は 0.5 以上ならやや影響強い。
#  1 を超えているデータは一度削除してからもう一度分析するのがよい
#  今回はシミュレーションデータを使ったので全く問題なし
par(mfrow=c(2,2))
plot(model.CPUE, which=1:4)
par(mfrow=c(1,1))

#  CPUE の標準化
# CPUE = Catch / Effort より、Effort＝1 の時の Catch が求まればよい
#  SST の影響を取り除くため、毎年 SST は平年値で一定と仮定する

```

```
new.CPUE  <-  data.frame(Year=2001:2010,  Effort=rep(1,10),  SST  =
mean(data5$SST))
new.CPUE

s.CPUE <- predict(model.CPUE, new.CPUE, type="respons")
s.CPUE
Mean.CPUE <- as.numeric( tapply(data5$CPUE, data5$Year, mean) )
Mean.CPUE

Y <- 2001:2010
Y
plot(Mean.CPUE ~ Y, type ="l", lwd=2,
  xlim = c(2001,2010), ylim=c(0, 2.5), main="CPUE 標準化")
lines(s.CPUE ~ Y, col=2, lwd=2)
legend("topleft", lwd=2, col=c(1,2),
  legend=c("年平均 CPUE","標準化された CPUE"))


# predict 関数を使わないで実装する
model.CPUE$coef
exp(model.CPUE$coef[1])

model.CPUE$coef


#  CPUE の年変動の「トレンド」を見るだけなら、SST の係数は使わなくても OK
s.CPUE.2 <- numeric()
s.CPUE.2[1] <- exp(model.CPUE$coef[1])
for ( i in 1:9){
       s.CPUE.2[i+1] <- exp(model.CPUE$coef[1] + model.CPUE$coef[i+1])
}
s.CPUE.2

#  全く同じトレンドになっていることに注目
par(mfrow=c(2,1))
plot(s.CPUE ~ Y, type="l")
```

```
plot(s.CPUE.2 ~ Y, type="l")
par(mfrow=c(1,1))


# ♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪
# ♪          写経終わり              ♪
# ♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪


#  SAS（別の統計ソフト）のような出力を出すなら options で設定する
#  R に慣れるため、あまり使わないほうが良い？
options("contrasts"=c("contr.SAS","contr.SAS"))
model.CPUE.SAS <- glm(
  Catch ~ as.factor(Year) + offset(log(Effort)) + SST,
  family="poisson",
  data=data5)
summary(model.CPUE.SAS)$coefficient

# 両者の比較
summary(model.CPUE)$coef
summary(model.CPUE.SAS)$coef

#  係数の見方
#  無理に SAS の表記を使う必要はないと思うが、水産ではなぜかつかわれているみたい

# 2001 年
model.CPUE$coef[1]
model.CPUE.SAS$coef[1] + model.CPUE.SAS$coef[2]
#  2002 年
model.CPUE$coef[1]  + model.CPUE$coef[2]
model.CPUE.SAS$coef[1] + model.CPUE.SAS$coef[3]

# 2010 年（最後の年）の SAS の出力はなぜか切片の値になっている。
model.CPUE$coef[1]  + model.CPUE$coef[10]
model.CPUE.SAS$coef[1]
```

```
793
794  #  predict 関数を使わないで、標準化する
795  exp(model.CPUE$coef[1] + mean(data5$SST) * model.CPUE$coef[11])
796  exp(model.CPUE$coef[1] + model.CPUE$coef[2] +
797    mean(data5$SST) * model.CPUE$coef[11])
798  exp(model.CPUE$coef[1] + model.CPUE$coef[3] +
799    mean(data5$SST) * model.CPUE$coef[11])
800  exp(model.CPUE$coef[1] + model.CPUE$coef[4] +
801    mean(data5$SST) * model.CPUE$coef[11])
802  exp(model.CPUE$coef[1] + model.CPUE$coef[5] +
803    mean(data5$SST) * model.CPUE$coef[11])
804  exp(model.CPUE$coef[1] + model.CPUE$coef[6] +
805    mean(data5$SST) * model.CPUE$coef[11])
806
807  s.CPUE
808
809  #  CPUE の年変動の「トレンド」を見るだけなら、SST の係数は使わなくても OK
810  s.CPUE.2 <- numeric()
811  s.CPUE.2[1] <- exp(model.CPUE$coef[1])
812  for ( i in 1:9){
813    s.CPUE.2[i+1] <- exp(model.CPUE$coef[1] + model.CPUE$coef[i+1])
814  }
815  s.CPUE.2
816
817  #  全く同じトレンドになっていることに注目
818  par(mfrow=c(2,1))
819  plot(s.CPUE ~ Y, type="l")
820  plot(s.CPUE.2 ~ Y, type="l")
821  par(mfrow=c(1,1))
822
823  #  単に SST の分をかけているだけ。
824  #  一般に「標準化 CPUE」を出す際に、predict の結果を使っているのか、係数の年変動
825  を使っているのかは不明
826  s.CPUE / s.CPUE.2
827  exp(mean(data5$SST) * model.CPUE$coef[11])
828
```

```
# SAS の出力を用いて CPUE 標準化
model.CPUE.SAS$coef
s.CPUE.2.SAS <- numeric()
for ( i in 1:9){
  s.CPUE.2.SAS[i] <- exp(model.CPUE.SAS$coef[i+1])
}
# 最終年を 0 に合わせているのでこうなる。
#  ややわかりにくい。
s.CPUE.2.SAS[10] <- exp(0)

Mean.CPUE
s.CPUE
s.CPUE.2
s.CPUE.2.SAS

s.CPUE / s.CPUE.2
s.CPUE / s.CPUE.2.SAS

# 倍数の意味
exp(mean(data5$SST) * model.CPUE$coef[11])
exp(mean(data5$SST) * model.CPUE.SAS$coef[11] +
  model.CPUE.SAS$coef[1])
exp(model.CPUE.SAS$coef[1])

mean(data5$Catch)
head(data5)

#  トレンドはどれを使っても同じ
par(mfrow=c(2,2))
plot(Mean.CPUE ~ Y, type="l")
plot(s.CPUE ~ Y, type="l")
plot(s.CPUE.2 ~ Y, type="l")
plot(s.CPUE.2.SAS ~ Y, type="l")
par(mfrow=c(1,1))
```

```
#=========================================
# ロジスティック回帰
#  Ｒにもともと入っているタイタニックデータを使用
#=========================================


# ★★★★★★★★★★★★★★★★★★
# ★          写経はじめ          ★
# ★★★★★★★★★★★★★★★★★★


Titanic

#  データの整形。使いにくいので
Titanic2 <- data.frame(Titanic)
Titanic2

data.Titanic <- data.frame(
  cbind(Titanic2[1:16,-4], Survive = Titanic2[17:32,5]))
names(data.Titanic)[4] <- "Death"
data.Titanic

#  二項分布のモデルの作成
# cbind(成功事例, 失敗事例)
model.binom_1 <- glm(
  cbind(Survive, Death) ~ .,
  data=data.Titanic,
  family="binomial")
summary(model.binom_1)

#  オッズの変化率
exp(coef(model.binom_1))[-1]

# モデルの診断
par(mfrow=c(2,2))
plot(model.binom_1, which=1:4)
```

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


# 過分散しているので、擬似尤度を使う
#  本当は後で説明するGLMMを使う方がよい
# なぜならばAICが計算できないから
#  擬似尤度なので、本物の尤度が計算できない
#  ということは　対数尤度-パラメタ数で計算するAICももちろん計算できない。やや不便だが、検定しか使わないならべつにこれでもよい。
model.binom_2 <- glm(
  cbind(Survive, Death) ~ .,
  data=data.Titanic,
  family="quasibinomial")
summary(model.binom_2)
exp(coef(model.binom_2))

#  擬似尤度を使う場合はF検定になる
#  だって尤度が計算できてないんだから
Anova(model.binom_2, test="F")


#  AICのけっか
#  検定と同じ
library(MuMIn)
dredge(model.binom_1)
# 擬似尤度のものは無理
dredge(model.binom_2)


# ♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪
# ♪　　写経終わり　　♪
# ♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪
```

```
#=====================
# Gammma 分布
#=====================

# ★★★★★★★★★★★★★★★★★★★
# ★          写経はじめ            ★
# ★★★★★★★★★★★★★★★★★★★

cars
plot(cars)

#  ガンマ回帰
model.Gamma <- glm(dist ~ speed, data=cars, family="Gamma"(link=log))
summary(model.Gamma)

# モデルの診断
par(mfrow=c(2,2))
plot(model.Gamma, which=1:4)
par(mfrow=c(1,1))

# ♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪
# ♪          写経終わり            ♪
# ♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪

# 予測区間を出す

new <- data.frame(
  speed = seq(min(cars$speed), max(cars$speed), by = 0.1))
new

pred.Gamma.response <- predict(model.Gamma, new, type="response")
pred.Gamma.response


# 必要なパラメタの準備
library("MASS")
```

```
shape <- gamma.shape(model.Gamma)$alpha
shape
scale <- pred.Gamma.response/shape # scale パラメータ
lower <- qgamma(0.025, shape = shape, scale = scale)
upper <- qgamma(0.975, shape = shape, scale = scale)


#  予測区間の図示
par(mfrow=c(1,2))
plot(cars, ylim =c(-50,120), main="Gamma 分布", cex.main=2)
lines(new$speed, pred.Gamma.response, lwd=2)
lines(new$speed, lower, lty = 2, lwd=2, col=4)
lines(new$speed, upper, lty = 2, lwd=2, col=2)
abline(h=0)

# 比較用の普通の正規線形モデル（単回帰）
model.lm <- lm(dist ~ speed, data=cars)
pred.lm <- predict(model.lm, new, interval="prediction")
head(pred.lm)
plot(cars, ylim =c(-50,120), main="正規分布", cex.main=2)
lines(new$speed, pred.lm[,1], lwd=2)
lines(new$speed, pred.lm[,2], lty = 2, lwd=2, col=4)
lines(new$speed, pred.lm[,3], lty = 2, lwd=2, col=2)
abline(h=0)
par(mfrow=c(1,1))


```

```
998  #======================
999  # 対数線形モデル
1000 #======================
1001
1002 # ★★★★★★★★★★★★★★★★★★★
1003 # ★          写経はじめ            ★
1004 # ★★★★★★★★★★★★★★★★★★★
1005
1006
1007 sample.table <- data.frame(
1008   type = rep(c("type_A","type_B"), 2),
1009   Result = rep(c("Good","Bad"), each=2),
1010   Freq = c(35,45,50,30)
1011 )
1012 sample.table
1013
1014 sample.table2 <- xtabs(Freq ~., sample.table)
1015 sample.table2
1016
1017 # χ二乗検定
1018 # イエーツの補正をされているらしい？
1019 chisq.test(sample.table2)
1020 # 普通のχ二乗検定
1021 chisq.test(sample.table2, correct=F)
1022 #  フィッシャーの正確確率検定
1023 fisher.test(sample.table2)
1024
1025 #  対数線形モデル
1026 #  交互作用の有無を検定する
1027 glm.loglin <- glm(Freq ~ (.)^2, data=sample.table, family="poisson")
1028 summary(glm.loglin)
1029 library(car)
1030 Anova(glm.loglin, type="III")
1031
1032 #  loglin 関数を使う
1033 m.loglin <- loglin(sample.table2, margin=list(c(1), c(2)), param=T,
```

```
fit=F)
summary(m.loglin)

#  対数尤度比統計量(deviance の差)
m.loglin$lrt

#  尤度比検定
1 - pchisq(m.loglin$lrt, m.loglin$df)

#  χ二乗検定
m.loglin$pearson
1 - pchisq(m.loglin$pearson, m.loglin$df)
chisq.test(sample.table2, correct=F)



# もっと複雑なデータ
HairEyeColor
# ポアソン回帰できるようにデータを整形
Hair <- data.frame(HairEyeColor)
Hair

#対数線形モデル
model.loglin.full <- glm(Freq ~ (.)^3, data=Hair, family="poisson")

#  検定
summary(model.loglin.full)
Anova(model.loglin.full, type="III")

# loglin を使っても一緒の結果。
#  margin の設定が難しいので、glm 関数のほうが楽
fm <- loglin(HairEyeColor, list(c(1, 2), c(1, 3), c(2, 3)))
fm
1 - pchisq(fm$lrt, fm$df)
```

```
1070
1071  #  3 次の交互作用を切った
1072  model.loglin.2 <- update(model.loglin.full, ~. - Hair:Eye:Sex)
1073  Anova(model.loglin.2, type="III")
1074
1075  #  2 次の交互作用を一つ切った
1076  model.loglin.3 <- update(model.loglin.2, ~. - Eye:Sex)
1077  Anova(model.loglin.3, type="III")
1078  summary(model.loglin.3)
1079
1080  # モデルの診断
1081  par(mfrow=c(2,2))
1082  plot(model.loglin.3, which=1:4)
1083  par(mfrow=c(1,1))
1084
1085
1086  # AIC でモデル選択
1087  library(MuMIn)
1088  dredge(model.loglin.full, rank="AIC")
1089  plot(dredge(model.loglin.full, rank="AIC"))
1090
1091
1092  # ♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪
1093  # ♪          写経終わり              ♪
1094  # ♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪
1095
```

```
#=========================
# ゼロ切断とゼロ過剰モデル
#=========================

# ★★★★★★★★★★★★★★★★★★★
# ★          写経はじめ          ★
# ★★★★★★★★★★★★★★★★★★★

# install.packages("VGAM")
# install.packages("pscl")
library(VGAM)
library(pscl)

# Zero Truncated

# サンプルデータの作成
#  期待値を表すパラメタλの政界値は「1」
set.seed(1)
Y <- rpois(lambda=1, n=30)
Y
mean(Y)

#  問題なし
exp(glm(Y ~ 1,  family="poisson")$coef)


# ゼロを切断してみる
Y.cut <- subset(Y, Y>0)
Y.cut

# 期待値が高くなっている
# こちらはウソの期待値。これに騙されてはいけない。
mean(Y.cut)

#  普通の GLM
model.pois <- glm(Y.cut ~ 1,  family="poisson")
```

```
#  ゼロ切断モデル
model.pos.pois <- vglm(Y.cut ~ 1,  family="pospoisson")

#  係数の確認
summary(model.pois)
summary(model.pos.pois)

#  正しい期待値を推定出来ているのはゼロ切断モデル
mean(Y.cut)
mean(Y)
exp(coef(model.pois))
exp(coef(model.pos.pois))


# ♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪
# ♪         写経終わり            ♪
# ♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪

# 要因付きのモデル
set.seed(1)
N <- 200
x <- rnorm(n=N)
log.lambda <- x * 1.3 - 0.5
Y2 <- numeric()
for(i in 1:N) {
  Y2[i] <- rpois(lambda=exp(log.lambda[i]), n=1)
}

plot(Y2 ~ x)

#  普通の GLM で楽々計算できる
glm(Y2 ~ x, family="poisson")

#  サンプルデータをデータフレームに入れる
data <- data.frame(Y2 = Y2, x=x)
data
```

```
# ゼロを切った
data.cut <- subset(data, data$Y2>0)
data.cut

# 0 を切る前と切った後で比較
par(mfrow=c(1,2))
plot(data$Y2 ~ data$x, ylim=c(0,25))
plot(data.cut$Y2 ~ data.cut$x, ylim=c(0,25))
par(mfrow=c(1,1))

#  普通の GLM
model.pois_2 <- glm(Y2 ~ x, data=data.cut, family="poisson")
summary(model.pois_2)

#  ゼロ切断モデル
model.pos.pois_2 <- vglm(Y2 ~ x, data=data.cut, family="pospoisson")
summary(model.pos.pois_2)

#  係数の確認
coef(model.pois_2)
coef(model.pos.pois_2)
#  正しい係数
coef(glm(Y2 ~ x, family="poisson"))


# 残念ながら尤度比検定はちょっとやりにくい
library(car)
Anova(model.pos.pois_2)

#  lrtest を使う
lrtest(model.pos.pois_2)

# 残差の確認もやや面倒
plot(model.pos.pois_2)
```

```
#  一応出せるが、Working の残差。deviance ではない
plot(resid(model.pos.pois_2) ~ exp(predict(model.pos.pois_2)))


#  予測値の確認
plot(data.cut$Y2 ~ data.cut$x, ylim=c(0,25))

new <- data.frame(x = seq(-2,3, by=0.1))

lines (
  predict(glm(Y2 ~ x, family="poisson", data=data),
  new, type="respons") ~
  new$x, lwd=3)
lines (
  predict(model.pos.pois_2, new, type="respons") ~ new$x, lwd=2, col=2)
lines (
  predict(model.pois_2, new, type="respons") ~ new$x, lwd=2, col=4)

legend("topleft", lwd=c(3,2,2), col=c(1,2,4),
  legend=c("正しい","ゼロ切断","ダメな普通の GLM"))



#==================================
# ゼロ過剰モデル
#==================================

# ★★★★★★★★★★★★★★★★★★
# ★          写経はじめ          ★
# ★★★★★★★★★★★★★★★★★★

library(pscl)
library(lmtest)


#  データ
```

```
data("bioChemists", package = "pscl")
hist(bioChemists$art)
summary(bioChemists)
pairs(bioChemists)

# art : Dr の過去 3 年間の論文数
#  fem  :  性別
#  mar  :  結婚の有無
#  kid5  :  5 歳以下の子供の数
#  phd  :  Dr の部門の信望(?)  prestige of Ph.D. department
#  ment  :  担当教官の過去 3 年間の論文数


# 普通の GLM

#  ポアソン分布と擬似尤度
model_pois <- glm(art ~ ., data = bioChemists, family = poisson)
model_qpois <- glm(art ~ ., data = bioChemists, family = quasipoisson)

#  負の二項分布
library(MASS)
model_nb <- glm.nb(art ~ ., data = bioChemists)

summary(model_pois)
summary(model_qpois)
summary(model_nb)

#  ネストしていないけれども、こんなことされる場合もある
#  実用上はそれほど問題ないらしい？
AIC(model_pois,model_nb)

# ♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪
# ♪          写経終わり              ♪
# ♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪

# 二項分布の部分には変数を入れず、定数だけにした Zero－Inflated モデル
```

```
#  art ~ .  の[.]は全ての変数を入れるという指示。「1」を入れると、定数項だけになる。

model_zip <- zeroinfl(art ~ . | 1, data = bioChemists)
model_zinb <- zeroinfl(art ~ . | 1, data = bioChemists, dist = "negbin")
summary(model_zip)
summary(model_zinb)

AIC(model_zip,model_zinb)          #  負の二項分布を支持

#  しかし、これだとセロインフルモデルにした意味がない
summary(model_zinb)
summary(model_nb)

#  MuMIn によるモデル選択にも対応している
library(MuMIn)
list.1 <- dredge(model_zinb, rank="AIC")
list.1

plot(list.1)

all.model <- get.models(list.1)
best.model<-all.model[1]
best.model



# ★★★★★★★★★★★★★★★★★★
# ★          写経はじめ            ★
# ★★★★★★★★★★★★★★★★★★


# 全部入れたモデル
#  理由は分からないが、略式で formula を書くと、MuMIn を使った時にバグが出る
model_zip2 <- zeroinfl(
  art ~ fem + mar + kid5 + phd + ment |
```

```
        fem + mar + kid5 + phd + ment, data = bioChemists)
model_zinb2 <- zeroinfl(
  art ~ fem + mar + kid5 + phd + ment |
        fem + mar + kid5 + phd + ment, data = bioChemists, dist = "negbin")

AIC(model_zip2,model_zinb2)

# ZINB は二項分布における正負の符号が違っていることに注意
#  なぜならば、ZINB は「偽物のゼロ」であるかどうかを二項分布で見分けているから。
#  プラスの影響ならば「偽物のゼロ」が生じやすいということ

summary(model_zinb2)


# 大変時間がかかるので注意
#  やらない方がよいかも
# 7 分くらいかかった
#  また、分離が起こって、計算できないモデルも多かった

library(MuMIn)
#  Not Run
# system.time( list.2 <- dredge(model_zinb2, rank="AIC"))

plot(list.2)
head(list.2, 10)

# all.model.2 <- get.models(list.2)
# best.model.2 <- all.model.2[1]
best.model.2

#  先生がたくさん論文を書いていると、学生が優秀なのにうっかりで
#  論文を出さなくなっちゃうことは少ない。ゆえに影響力は負。
#  結婚していると、学生が優秀なのにうっかりで論文を出さなくなっちゃうことは少ない。
#  ゆえに影響力は負。
#  やる気が増すのかもしれない？
# 変数の解釈が難しいので要注意。
```

```
1348
1349
1350  # 計算できていない変数の組み合わせも多数あることに注意
1351  #  手作業で検定するのが最も安全かもしれない。ただし、手間はかかる。
1352  warnings()
1353
1354
1355  # hurdle モデル
1356  #  ゼロインフルとはやや異なる結果になるが、どちらを選ぶかは難しい
1357  # 全部入れたモデル
1358  model_hurdle <- hurdle(
1359    art ~ fem + mar + kid5 + phd + ment |
1360          fem + mar + kid5 + phd + ment,
1361    data = bioChemists, dist = "negbin")
1362
1363  #  二項分布にプラスの影響ならば純粋に「論文を投稿しやすい」という意味になる
1364  summary(model_hurdle)
1365  summary(model_zinb2)
1366
1367
1368
1369  # ♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪
1370  # ♪          写経終わり              ♪
1371  # ♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪
1372
1373
1374  # 先生の論文数と学生が論文を「書くかどうか」の関係性
1375
1376  head(bioChemists)
1377  bioChemists.bin1 <- cbind(subset(bioChemists, art == 0), bin = 0)
1378  bioChemists.bin2 <- cbind(subset(bioChemists, art != 0), bin = 1)
1379  bioChemists.bin <- rbind(bioChemists.bin1, bioChemists.bin2)
1380  bioChemists.bin
1381
1382  model_bin <- glm(
1383    bin ~ fem + mar + kid5 + phd + ment,
```

```
  data=bioChemists.bin,
  family="binomial")
summary(model_bin)


#  書いた人はどれだけたくさん書くか。
bioChemists.pos <- subset(bioChemists, art != 0)
bioChemists.pos

library(VGAM)
model.pos <- vglm(
  art ~ fem + mar + kid5 + phd + ment,
  family="posnegbinomial",
  data=bioChemists.pos)
summary(model.pos)


# ZINB は二項分布における正負の符号が違っていることに注意
#  なぜならば、ZINB は「偽物のゼロ」であるかどうかを二項分布で見分けているから。
#  プラスの影響ならば「偽物のゼロ」が生じやすいということ
#  先生がたくさん論文を書いていると、学生が優秀なのにうっかりで論文を出さなくなっちゃうことは少ない。ゆえに影響力は負。

# 通常の GLM と比較すると hurdle は、二項分布＋ゼロ切断だということがよくわかる
#  二項分布にプラスの影響ならば純粋に「論文を投稿しやすい」という意味になる
#  先生がたくさん論文を投稿していたら、学生も論文を投稿しやすいという意味
summary(model_zinb2)
summary(model_hurdle)
summary(model_bin)

summary(model.pos)
summary(model_hurdle)
```

```
#====================
# GLMM
#====================

# ★★★★★★★★★★★★★★★★★★★
# ★          写経はじめ            ★
# ★★★★★★★★★★★★★★★★★★★


#=========================
#  Step1  過分散への対応
#=========================
data6 <- read.csv("data6.csv")

head(data6)
pairs(data6)

#  通常の GLM
model.glm.pois <- glm(
  y ~ x1 + x2,
  family="poisson",
  data=data6)
summary(model.glm.pois)

#  尤度比検定  Type II ANOVA
library(car)
Anova(model.glm.pois)

# 擬似尤度
model.glm.quasi.pois <- glm(
  y ~ x1 + x2,
  family="quasipoisson",
  data=data6)
summary(model.glm.quasi.pois)


```

```
# 尤度比検定
Anova(model.glm.quasi.pois)

#  GLMM
library(lme4)

# ランダムエフェクトを入れる準備
ID <- 1:nrow(data6)
ID

data6$ID <- ID
head(data6)

#  GLMM
model.glmm <- glmer(
  y ~ x1 + x2 + (1 | ID),
  data=data6,
  family=poisson)
summary(model.glmm)
Anova(model.glmm)

# 比較
summary(model.glm.pois)
summary(model.glm.quasi.pois)
summary(model.glmm)



#=========================
#  Step2.擬似反復を攻略する
#=========================
head(sleepstudy)
pairs(sleepstudy)

# 睡眠時間が減ることによる悪影響を調べた実験データ
#  初日は普通に寝る
```

```
#  二日目以降は一日 3 時間しか寝ない
#  Reaction はあるテストに対する反応速度。早いほうが良い評価である。
#  寝不足は反応速度にどれくらい影響するか?

# このデータは一人の被験者につき 10 日間調べている
#  本当のサンプルサイズは被験者の数であるはず
#  しかし、GLM をした時のサンプルサイズは実質 10 倍されている
#  サンプルサイズが多いように見せかけられると有意になりやすくなってしまう。

glm.sleep <- glm(Reaction ~ Days, data=sleepstudy)
summary(glm.sleep)
Anova(glm.sleep)

# GLMM で擬似反復を攻略する

# 被験者ごとに反応速度が異なるというランダムエフェクトを入れる
glmm.sleep_1 <- lmer(Reaction ~ Days + (1|Subject), sleepstudy)

#  さらに追加で、被験者ごとに日数による影響が異なることもモデルに組み込む
#  0+がないと切片に 2 重にランダムエフェクトが組み込まれてしまうので注意
glmm.sleep_2 <- lmer(
  Reaction ~ Days + (1|Subject) + (0+Days|Subject), sleepstudy)

summary(glmm.sleep_1)
summary(glmm.sleep_2)

Anova(glmm.sleep_1)
Anova(glmm.sleep_2)

anova(glmm.sleep_1, glmm.sleep_2)


#  擬似反復と過分散を両方退治する

head(cbpp)
pairs(cbpp)
```

```
# herd 群れ番号
#  incidence  発症数
#  size  群れサイズ
#  period  期間
# period ごとの発症数の違いを見たい



# 擬似反復だけを取り除く
glmm.bin_1 <- glmer(
  cbind(incidence, size - incidence) ~ period + (1 | herd),
  family = binomial, data = cbpp)

summary(glmm.bin_1)

# さらに、過分散も考慮する
cbpp$ID <- 1:nrow(cbpp)
cbpp$ID
glmm.bin_2 <- glmer(
  cbind(incidence, size - incidence) ~ period + (1 | herd) + (1|ID),
  family = binomial,
  data = cbpp)

summary(glmm.bin_2)

# 二つのモデルの比較
anova(glmm.bin_1, glmm.bin_2)
AIC(glmm.bin_1,glmm.bin_2)


# ♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪
# ♪         写経終わり            ♪
# ♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪

# お疲れ様でした
```