GCS analysis

Bart Buelens, Rafael S. de Souza

December 8, 2014

This document contains some preliminary work possibly of use to a paper by the COIN group on negative binomial distributions to model count data.

It is put together in RStudio as a Rmd file from which a PDF is easily generated.

Introduction

The catalog of Globular Cluster Systems (GCS) presented in Harris et al. 2013 is analyzed. In particular, three statistical models relating the number of GCSs to the dynamical mass of galaxies are studied. These include a linear model assuming gaussianity, and generalized linear models assuming poisson and negative binomial distributions. At the end the analysis is repeated using the black hole mass instead of the dynamical mass.

Data

The data set 'GCs.csv' sent by e-mail by Rafael on Dec 1st is used.

```
GCS = read.csv(file="..//Dataset//GCs.csv",header=TRUE,dec=".",sep="")
GCS = subset(GCS, !is.na(Mdyn)) # 1 removed
dim(GCS)
```

```
## [1] 45 15
```

Of the initial 422 galaxies in the catalog, 45 are retained for which variables needed here are present. The errors are used further down. Weights are defined to be the inverse of the error.

```
GCS$w = 1/ GCS$N_GC_err^2
```

It is somewhat strange that in the catalogue the GCS count is not always integer; an integer version is derived here simply by rounding. * NOTE: in Rafas file these are integer! *

Before proceding some useful libraries are loaded.

```
library(ggplot2)
library(MASS)     # glm.nb
library(COUNT)   # diagnostics
library(lmtest) # Likelihood test
library(gamlss.tr)# zero truncated model
#library(glmmADMB)  # GLMM
library(caret) # 10-fold
library(reshape) # melt
library(scales)
require(ggthemes)
```

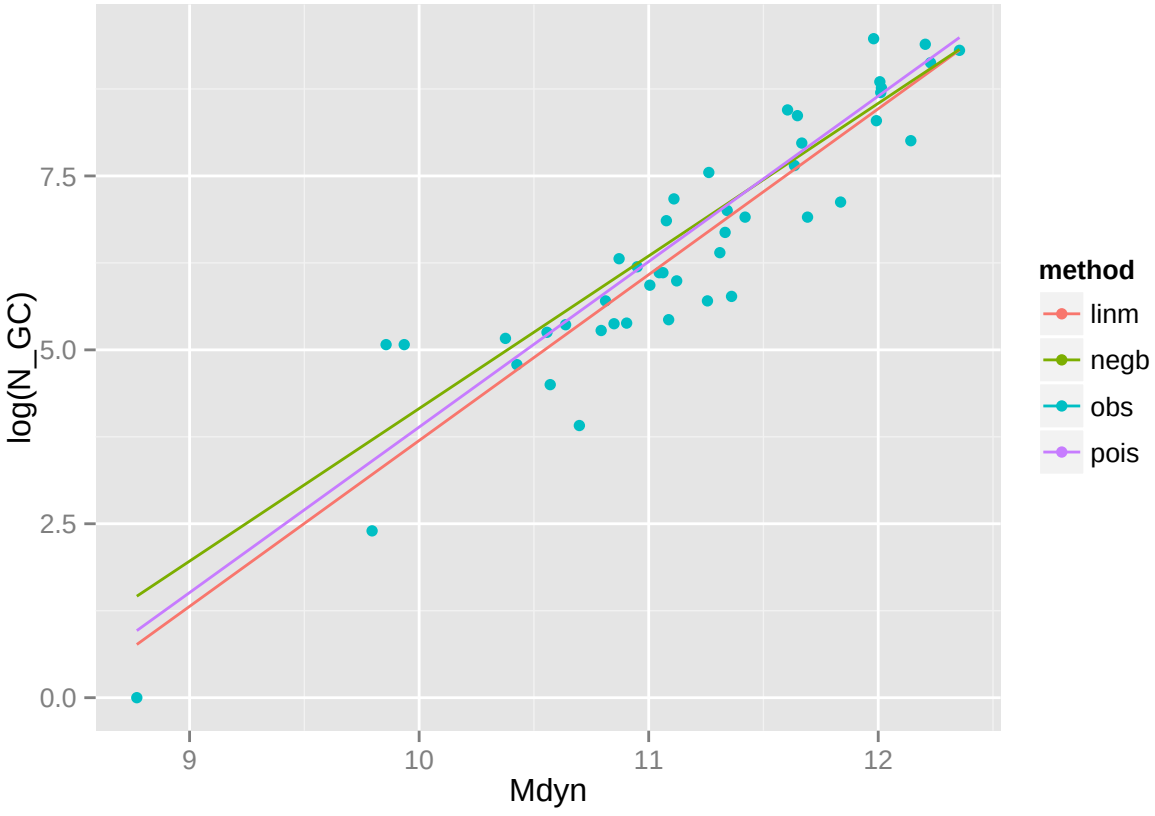
Modeling

The first way to model counts is using a linear model for the log counts. The second is a glm poisson model, the third is a glm with negative binomial. These are specified and fitted as follows.

```
linmFit = glm(log(N_GC) ~ Mdyn, family="gaussian",GCS)
poisFit = glm(N_GC ~ Mdyn, GCS, family="poisson")
negbFit = glm.nb(N_GC ~ Mdyn, GCS)
```

The results are now plotted using ggplot() from the ggplot2 package.

```
X = GCS[,c("Galaxy","Mdyn","N_GC")]
X = rbind(X,X,X,X)
X$method = factor(rep(c("obs","linm","pois","negb"),each=nrow(GCS)))
X[X$method == "linm","N_GC"] = exp(linmFit$fitted.values)
X[X$method == "pois","N_GC"] = poisFit$fitted.values
X[X$method == "negb","N_GC"] = negbFit$fitted.values
ggplot(X, aes(x = Mdyn,y=log(N_GC))) +
   geom_point(aes(colour=method), data = subset(X, method=="obs")) +
   geom_line(aes(colour=method), data = subset(X, method != "obs"))
```



The three are not dramatically different.

Model comparison

A quantitative comparison is obtained using leave-one-out cross validation (LOO-CV). Each observation is left out once and predicted using a model fit on the other data. The errors are the differences between the

predictions and true values. The squared differences are weighted using the weights derived earlier, inversely proportional to the errors on the known values. LOO-CV is implemented as follows.

```
N = nrow(GCS)
A = data.frame(linm=rep(NA,N))
A$negb = A$pois = NA
for (i in 1:N) {
   Gx = GCS[-i,]
   f1 = lm(log(N_GC) ~ Mdyn, Gx)
   f2 = glm(N_GC ~ Mdyn, Gx, family="poisson")
   f3 = glm.nb(N_GC ~ Mdyn, Gx)
   A[i,"linm"] = GCS$w[i] * (GCS$N_GC[i] - exp(predict(f1, GCS[i,])))^2
   A[i, "pois"] = GCS$w[i] * (GCS$N_GC[i] - predict(f2, GCS[i,]))^2
   A[i, "negb"] = GCS$w[i] * (GCS$N_GC[i] - predict(f3, GCS[i,]))^2
}
```

The square root of the mean of the values in A gives the LOO-CV measure.

```
lapply(A, function(x) sqrt(mean(x)))
```

```
## $linm
## [1] 4.589
##
## $pois
## [1] 7.334
##
## $negb
## [1] 7.332
```

Based on the LOO-CV criterion, the linear model is best. In this case there is no benefit in using Poisson or negative binomial models.

Black hole mass

The same analysis now using the black hole mass as predictor rather than dynamical mass.

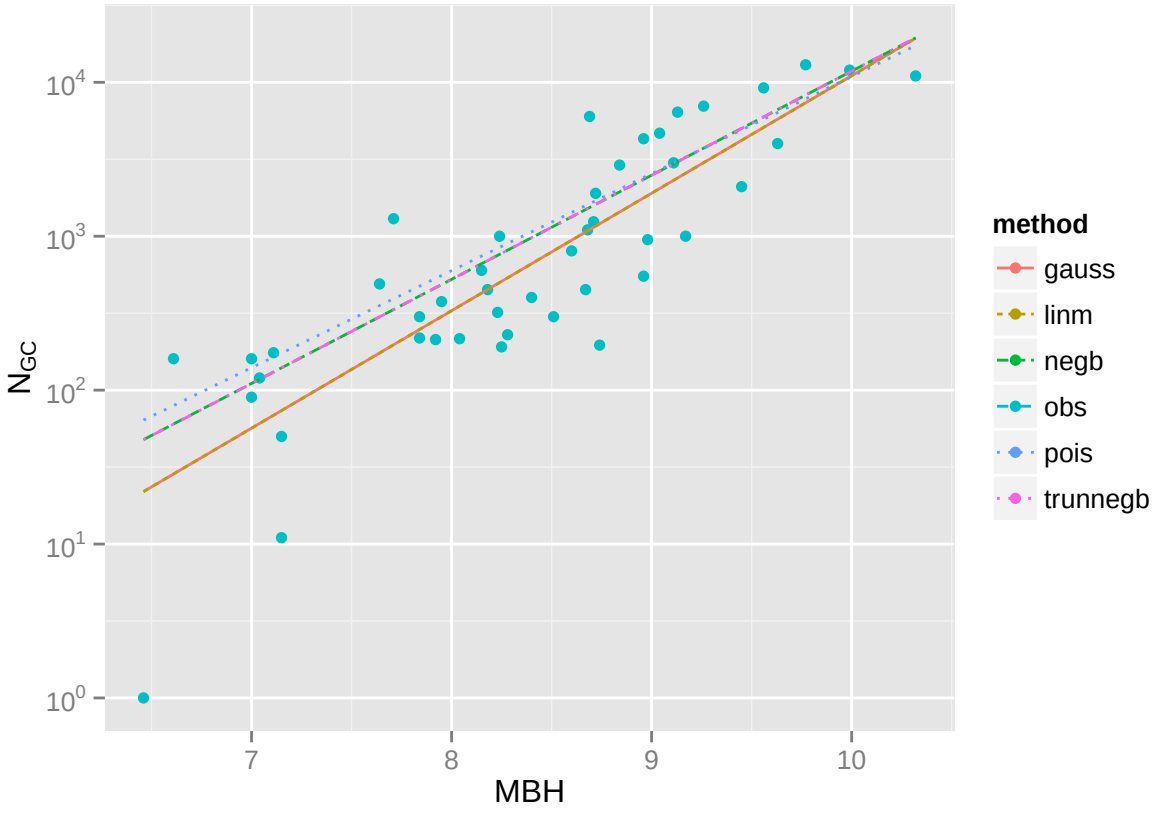
```
linmFit = lm(log(N_GC) ~ MBH, GCS)
gaussFit = glm(log(N_GC) ~ MBH, family="gaussian",GCS)
poisFit = glm(N_GC ~MBH, GCS, family="poisson")
negbFit = glm.nb(N_GC ~ MBH, GCS)
#negbmFit <- glmmadmb(N_GC~MBH +(1|Type),
#                     data=GCS, family="nbinom")
gen.trun(0,"NBI",type="left",name="lefttr")
```

```
## A truncated family of distributions from NBI has been generated
##  and saved under the names:
##  dNBIlefttr pNBIlefttr qNBIlefttr rNBIlefttr NBIlefttr
## The type of truncation is left and the truncation parameter is 0
```

```
trunFit=gamlss(N_GC ~ MBH, data=GCS[,c("N_GC","MBH")],family=NBIlefttr)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 705.4
## GAMLSS-RS iteration 2: Global Deviance = 705.2
## GAMLSS-RS iteration 3: Global Deviance = 705.2
```

```
X = GCS[,c("Galaxy","MBH","N_GC")]
X = rbind(X,X,X,X,X,X)
X$method = factor(rep(c("obs","linm","gauss","pois","negb","trunnegb"),each=nrow(GCS)))
X[X$method == "linm","N_GC"] = exp(linmFit$fitted.values)
X[X$method == "gauss","N_GC"] = exp(gaussFit$fitted.values)
X[X$method == "pois","N_GC"] = poisFit$fitted.values
X[X$method == "negb","N_GC"] = negbFit$fitted.values
X[X$method == "trunnegb","N_GC"] = predict(trunFit, type="response")
ggplot(X, aes(x = MBH,y=N_GC,linetype=method)) +
  scale_y_continuous(trans = 'log10',
                     breaks=trans_breaks("log10",function(x) 10^x),
                     labels=trans_format("log10",math_format(10^.x)))+
   geom_point(aes(colour=method), data = subset(X, method=="obs")) +
   geom_line(aes(colour=method), data = subset(X, method != "obs"))+ylab(expression(N[GC]))
```




```
A = data.frame(linm=rep(NA,N))
A$negb = A$pois = A$gauss= NA
for (i in 1:N) {
   Gx = GCS[-i,]
   f1 = lm(log(N_GC) ~ MBH, Gx)
```

```
   f2 = glm(log(N_GC) ~ MBH, family="gaussian",Gx)
   f3 = glm(N_GC ~ MBH, Gx, family="poisson")
   f4 = glm.nb(N_GC ~ MBH, Gx)
   A[i,"linm"] = GCS$w[i] * (GCS$N_GC[i] - exp(predict(f1, GCS[i,],type="response")))^2
   A[i,"gauss"] = GCS$w[i] * (GCS$N_GC[i] - exp(predict(f2, GCS[i,],type="response")))^2
   A[i, "pois"] = GCS$w[i] * (GCS$N_GC[i] - predict(f3, GCS[i,],type="response"))^2
   A[i, "negb"] = GCS$w[i] * (GCS$N_GC[i] - predict(f4, GCS[i,],type="response"))^2
}
lapply(A, function(x) sqrt(mean(x)))
```

```
## $linm
## [1] 6.731
##
## $gauss
## [1] 6.731
##
## $pois
## [1] 10.45
##
## $negb
## [1] 9.459
```

10-fold cross validation

```
folds <- createFolds(GCS$N_GC, k=10)
A = data.frame(linm=rep(NA,N))
A$negb = A$pois = A$gauss= NA
for (i in 1:10) {
   Gx = GCS[-folds[[i]],]
   f1 = lm(log(N_GC) ~ MBH, Gx)
   f2 = glm(log(N_GC) ~ MBH, family="gaussian",Gx)
   f3 = glm(N_GC ~ MBH, Gx, family="poisson")
   f4 = glm.nb(N_GC ~ MBH, Gx)
   A[folds[[i]],"linm"] = GCS$w[folds[[i]]] * (GCS$N_GC[folds[[i]]] - exp(predict(f1, GCS[folds[[i]],],
   A[folds[[i]],"gauss"] = GCS$w[folds[[i]]] * (GCS$N_GC[folds[[i]]] - exp(predict(f2, GCS[folds[[i]],]
   A[folds[[i]], "pois"] = GCS$w[folds[[i]]] * (GCS$N_GC[folds[[i]]] - predict(f3, GCS[folds[[i]],],typ
   A[folds[[i]], "negb"] = GCS$w[folds[[i]]] * (GCS$N_GC[folds[[i]]] - predict(f4, GCS[folds[[i]],],typ
}
lapply(A, function(x) sqrt(mean(x)))
```

```
## $linm
## [1] 6.896
##
## $gauss
## [1] 6.896
##
## $pois
## [1] 10.21
##
## $negb
## [1] 9.466
```

Errors for GLM Prediction

Confidence interval

```
pred_glm<-function(x){
preds <- predict(x, type = "link", se.fit = TRUE)
critval <- 1.96 ## approx 95% CI
upr <- preds$fit + (critval * preds$se.fit)
lwr <- preds$fit - (critval * preds$se.fit)
fit <- preds$fit

fit2 <- x$family$linkinv(fit)
upr2 <- x$family$linkinv(upr)
lwr2 <- x$family$linkinv(lwr)
errup<-upr2-fit2
errlow<-fit2-lwr2
return(list(fit=fit2,lwr=lwr2,upr=upr2,errlow =errlow,errup=errup))
}
```

Prediction

The above computes a confidence interval. Following code is an attempt to compute prediction intervals: taking into account variance of estimated parameters through bootstrap, and prediction uncertainty due to model variance.

```
N = nrow(GCS)
# predN = 50
# MBHx = data.frame(MBH = seq(from = 0.9 * min(GCS$MBH),
#                             to = 1.1 * max(GCS$MBH),
#                             length.out = predN))
# Do the following instead, to enable calculation of coverage.
MBHx = data.frame(MBH = GCS$MBH)
predN = length(GCS$MBH)
bN = 1000
Blinm = Bpois = Bnegb = matrix(data = NA, nrow = bN, ncol = predN)
BlinmU = BlinmL = BpoisU = BpoisL = BnegbU = BnegbL = Blinm
qlwr = 0.025
qupr = 0.975
for (i in 1:bN) {
  G = GCS[sample(1:N, N, replace = TRUE),]
  linmFit = glm(log(N_GC) ~ MBH, G, family="gaussian")
  poisFit = glm(N_GC ~ MBH, G, family="poisson")
  negbFit = glm.nb(N_GC ~ MBH, G)
  linmP = predict(linmFit, MBHx)
  poisP = predict(poisFit, MBHx)
  negbP = predict(negbFit, MBHx)
  Blinm[i,] = linmP
  BlinmL[i,] = qnorm(qlwr, mean = linmP, sd = sd(residuals(linmFit)))
  BlinmU[i,] = qnorm(qupr, mean = linmP, sd = sd(residuals(linmFit)))
  Bpois[i,] = poisP
  BpoisL[i,] = log(qpois(qlwr, lambda = exp(poisP)))
```

```
  BpoisU[i,] = log(qpois(qupr, lambda = exp(poisP)))
  Bnegb[i,] = negbP
  BnegbL[i,] = log(qnbinom(qlwr, size = negbFit$theta, mu = exp(negbP)))
  BnegbU[i,] = log(qnbinom(qupr, size = negbFit$theta, mu = exp(negbP)))
}

# fit once more, on the full sample, as a reference:
linmFit = glm(log(N_GC) ~ MBH, GCS, family="gaussian")
poisFit = glm(N_GC ~ MBH, GCS, family="poisson")
negbFit = glm.nb(N_GC ~ MBH, GCS)
linmPred = predict(linmFit, MBHx)
poisPred = predict(poisFit, MBHx)
negbPred = predict(negbFit, MBHx)

# Compare bootstrap means with reference to check validity of bootstrap:
summary(colMeans(Blinm) - linmPred)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -0.01040 -0.00496 -0.00306 -0.00258 -0.00034  0.00525
```

```
summary(colMeans(Bpois) - poisPred)
```

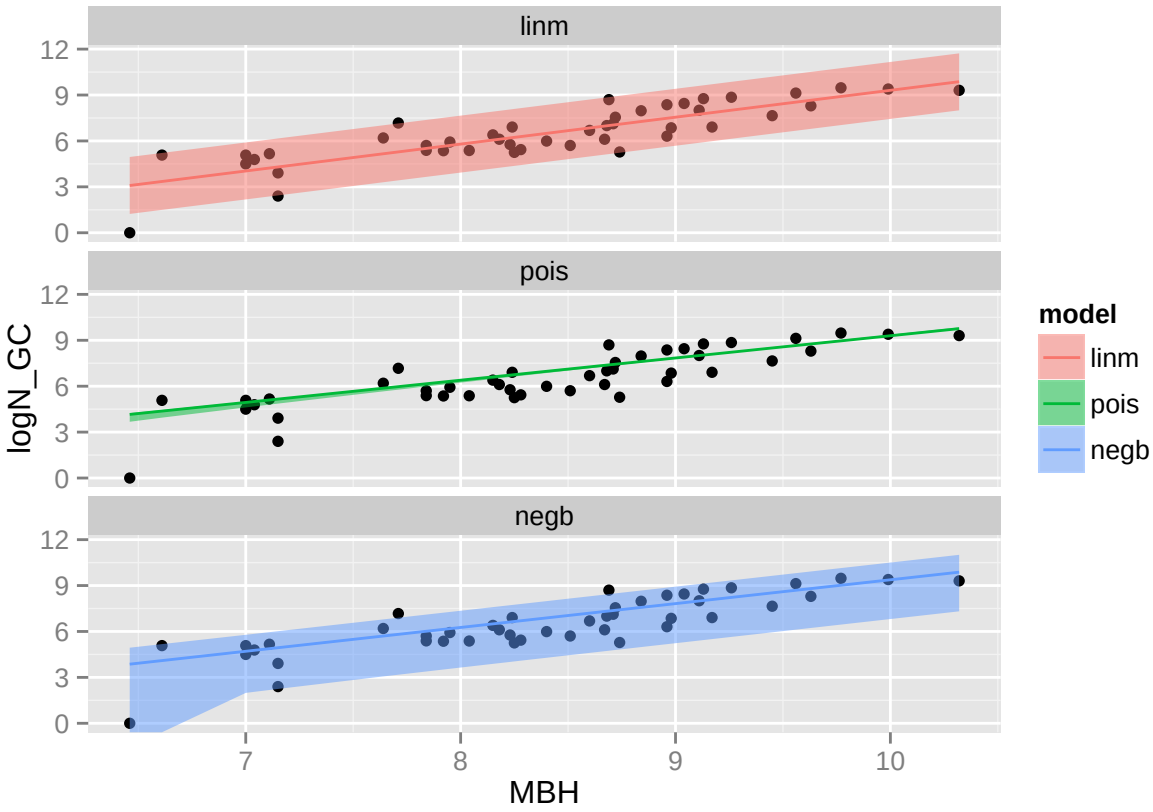
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.1650 -0.0811 -0.0404 -0.0475 -0.0119  0.0694
```

```
summary(colMeans(Bnegb) - negbPred)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.0562 -0.0316 -0.0197 -0.0218 -0.0113  0.0125
```

```
# Now plot predicted means and prediction intervals
P = MBHx
P$linm = linmPred
P$pois = poisPred
P$negb = negbPred
P = melt(P, id.vars = "MBH", variable_name = "model")
names(P)[3] = "logN_GC"
P$lwr = c(colMeans(BlinmL), colMeans(BpoisL), colMeans(BnegbL))
P$upr = c(colMeans(BlinmU), colMeans(BpoisU), colMeans(BnegbU))

GCS$logN_GC = log(GCS$N_GC)
ggplot(GCS, aes(x = MBH, y = logN_GC)) +
  geom_point() +
  geom_line(data=P, aes(x = MBH, y=logN_GC, colour=model)) +
  geom_ribbon(data=P, aes(ymin=lwr, ymax=upr, fill=model), alpha=0.5) +
  facet_wrap(~ model, ncol=1)
```



Consider coverage.

```
P$obs = rep(GCS$logN_GC,3)
P$cov = P$lwr <= P$obs & P$obs <= P$upr
table(P[,c("model","cov")])
```

```
##       cov
## model  FALSE TRUE
##   linm     3   42
##   pois    43    2
##   negb     2   43
```

```
aggregate(P$cov, by=list(P$model), FUN = function(x) sum(x)/length(GCS$MBH))
```

```
##   Group.1       x
## 1    linm 0.93333
## 2    pois 0.04444
## 3    negb 0.95556
```

Negative binomial has best coverage, as it should be 95%. Poisson is very bad, as it is unable to capture the overdispersion.

Bias diagnostic

Modeling log counts as a linear model with gaussian error tends to lead to biased results on the actual scale of the counts. Compare the total number of GCS in the sample with the total of the fitted values.

```
sum(GCS$N_GC)
```

```
## [1] 101132
```

```
sum(exp(linmFit$fitted.values))
```

```
## [1] 86068
```

```
sum(poisFit$fitted.values)
```

```
## [1] 101132
```

```
sum(negbFit$fitted.values)
```

```
## [1] 102783
```

We could compute errors on these in the above bootstrap routine.

Dispersion statistics

```
P_disp<-function(x){
  pr<-sum(residuals(x,type="pearson")^2)
  dispersion<-pr/x$df.residual
  cat("\n Pearson Chi2 = ",pr,
      "\n Dispersion = ",dispersion, "\n")
}
P_disp(poisFit)
```

```
##
##  Pearson Chi2 =  44654
##  Dispersion =  1038
```

```
P_disp(negbFit)
```

```
##
##  Pearson Chi2 =  49.85
##  Dispersion =  1.159
```

Goodness of fit

Deviance is asymptotically chi squared:

```
1 - pchisq(summary(gaussFit)$deviance,summary(gaussFit)$df.residual)
```

```
## [1] 0.4774
```

```r
1 - pchisq(summary(poisFit)$deviance,summary(poisFit)$df.residual)
```

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

```r
1 - pchisq(summary(negbFit)$deviance,summary(negbFit)$df.residual)
```

```
## [1] 0.2075
```

The Poisson model is not good. The other two are not rejected based on the deviance assessment. Note that the sample is rather small so care must be taken with asymptotics.

References

Harris, W.E., Harris, G.L.H., and Alessi, M. 2013, ApJ 772, 82.