A couple of weeks ago, I was inspired by a study to about a classic design issue that arises in cluster randomized trials: should we focus on the number of clusters or the size of those clusters? This trial, which is concerned with preventing opioid use disorder for at-risk patients in primary care clinics, has also motivated this second post, which concerns another important issue – over-dispersion.

**A count outcome**

In this study, one of the primary outcomes is the number of days of opioid use over a six-month follow-up period (to be recorded monthly by patient-report and aggregated for the six-month measure). While one might get away with assuming that the outcome is continuous, it really is not; it is a *count* outcome, and the possible range is 0 to 180. There are two related questions here – what model will be used to analyze the data once the study is complete? And, how should we generate simulated data to estimate the power of the study?

In this particular study, the randomization is at the physician level so that all patients in a particular physician practice will be in control or treatment. (For the purposes of simplification here, I am going to assume there is no treatment effect, so that all variation in the outcome is due to physicians and patients only.) One possibility is to assume the outcome \(Y\_{ij}\) for patient \(i\) in group \(j\) has a binomial distribution with 180 different “experiments” – every day we ask did the patient use opioids? – so that we say \(Y\_{ij} \sim Bin(180, \ p\_{ij})\).

**The probability parameter**

The key parameter here is \(p\_{ij}\), the probability that patient \(i\) (in group \(j\)) uses opioids on any given day. Given the binomial distribution, the number of days of opioid use we expect to observe for patient \(i\) is \(180p\_{ij}\). There are at least three ways to think about how to model this probability (though there are certainly more):

* \(p\_{ij} = p\): everyone shares the same probability The collection of all patients will represent a sample from \(Bin(180, p)\).
* \(p\_{ij} = p\_j\): the probability of the outcome is determined by the cluster or group alone. The data within the cluster will have a binomial distribution, but the collective data set will *not* have a strict binomial distribution and will be over-dispersed.
* \(p\_{ij}\) is unique for each individual. Once again the collective data are over-dispersed, potentially even more so.

**Modeling the outcome**

The correct model depends, of course, on the situation at hand. What data generation process fits what we expect to be the case? Hopefully, there are existing data to inform the likely model. If not, it may by most prudent to be conservative, which usually means assuming more variation (unique \(p\_{ij}\)) rather than less (\(p\_{ij} = p\)).

In the first case, the probability (and counts) can be estimated using a generalized linear model (GLM) with a binomial distribution. In the second, one solution (that I will show here) is a generalized linear mixed effects model (GLMM) with a binomial distribution and a group level random effect. In the third case, a GLMM with a negative a *negative binomial* distribution would be more likely to properly estimate the variation.

**Case 1: binomial distribution**

Even though there is no clustering effect in this first scenario, let’s assume there are clusters. Each individual will have a probability of 0.4 of using opioids on any given day (log odds = -0.405):

def <- defData(varname = "m", formula = 100, dist = "nonrandom", id = "cid")

defa <- defDataAdd(varname = "x", formula = -.405, variance = 180,

dist = "binomial", link = "logit")

Generate the data:

set.seed(5113373)

dc <- genData(200, def)

dd <- genCluster(dc, cLevelVar = "cid", numIndsVar = "m", level1ID = "id")

dd <- addColumns(defa, dd)

Here is a plot of 20 of the 100 groups:

dplot <- dd[cid %in% c(1:20)]

davg <- dplot[, .(avgx = mean(x)), keyby = cid]

ggplot(data=dplot, aes(y = x, x = factor(cid))) +

geom\_jitter(size = .5, color = "grey50", width = 0.2) +

geom\_point(data = davg, aes(y = avgx, x = factor(cid)),

shape = 21, fill = "firebrick3", size = 2) +

theme(panel.grid.major.y = element\_blank(),

panel.grid.minor.y = element\_blank(),

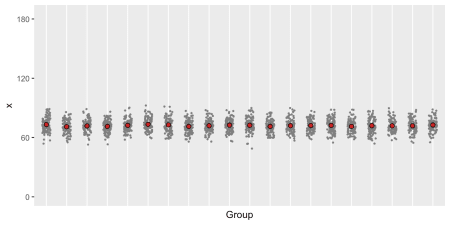
axis.ticks.x = element\_blank(),

axis.text.x = element\_blank()

) +

xlab("Group") +

scale\_y\_continuous(limits = c(0, 185), breaks = c(0, 60, 120, 180))



Looking at the plot, we can see that a mixed effects model is probably not relevant.

**Case 2: over-dispersion from clustering**

def <- defData(varname = "ceffect", formula = 0, variance = 0.08,

dist = "normal", id = "cid")

def <- defData(def, varname = "m", formula = "100", dist = "nonrandom")

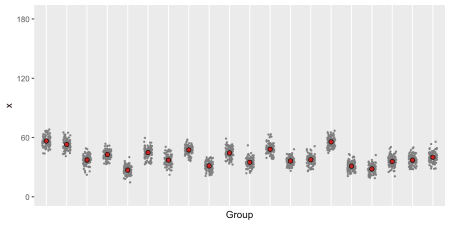
defa <- defDataAdd(varname = "x", formula = "-0.405 + ceffect",

variance = 100, dist = "binomial", link = "logit")

dc <- genData(200, def)

dd <- genCluster(dc, cLevelVar = "cid", numIndsVar = "m", level1ID = "id")

dd <- addColumns(defa, dd)



This plot suggests that variation *within* the groups is pretty consistent, though there is variation *across* the groups. This suggests that a binomial GLMM with a group level random effect would be appropriate.

**Case 3: added over-dispersion due to individual differences**

defa <- defDataAdd(varname = "ieffect", formula = 0,

variance = .25, dist = "normal")

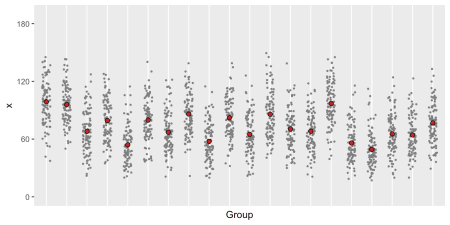
defa <- defDataAdd(defa, varname = "x",

formula = "-0.405 + ceffect + ieffect",

variance = 180, dist = "binomial", link = "logit")

dd <- genCluster(dc, cLevelVar = "cid", numIndsVar = "m", level1ID = "id")

dd <- addColumns(defa, dd)



In this last case, it is not obvious what model to use. Since there is variability within and between groups, it is probably safe to use a negative binomial model, which is most conservative.

**Estimating the parameters under a negative binomial assumption**

We can fit the data we just generated (with a 2-level mixed effects model) using a *single-level* mixed effects model with the assumption of a negative binomial distribution to estimate the parameters we can use for one last simulated data set. Here is the model fit:

nbfit <- glmer.nb(x ~ 1 + (1|cid), data = dd,

control = glmerControl(optimizer="bobyqa"))

broom::tidy(nbfit)

## # A tibble: 2 x 6

## term estimate std.error statistic p.value group

##

## 1 (Intercept) 4.29 0.0123 347. 0 fixed

## 2 sd\_(Intercept).cid 0.172 NA NA NA cid

And to generate the negative binomial data using simstudy, we need a dispersion parameter, which can be extracted from the estimated model:

(theta <- 1/getME(nbfit, "glmer.nb.theta"))

## [1] 0.079

revar <- lme4::getME(nbfit, name = "theta")^2

revar

## cid.(Intercept)

## 0.03

Generating the data from the estimated model allows us to see how well the negative binomial model fit the dispersed binomial data that we generated. A plot of the two data sets should look pretty similar, at least with respect to the distribution of the cluster means and within-cluster individual counts.

def <- defData(varname = "ceffect", formula = 0, variance = revar,

dist = "normal", id = "cid")

def <- defData(def, varname = "m", formula = "100", dist = "nonrandom")

defa <- defDataAdd(varname = "x", formula = "4.28 + ceffect",

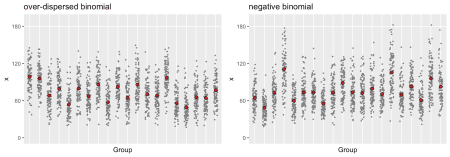
variance = theta, dist = "negBinomial", link = "log")

dc <- genData(200, def)

ddnb <- genCluster(dc, cLevelVar = "cid", numIndsVar = "m",

level1ID = "id")

ddnb <- addColumns(defa, ddnb)



The two data sets do look like they came from the same distribution. The one limitation of the negative binomial distribution is that the sample space is not limited to numbers between 0 and 180; in fact, the sample space is all non-negative integers. For at least two clusters shown, there are some individuals with counts that exceed 180 days, which of course is impossible. Because of this, it might be safer to use the over-dispersed binomial data as the generating process for a power calculation, but it would be totally fine to use the negative binomial model as the analysis model (in both the power calculation and the actual data analysis).

**Estimating power**

One could verify that power is indeed reduced as we move from *Case 1* to *Case 3*. (I’ll leave that as an exercise for you – I think I’ve provided many examples in the past on how one might go about doing this. If, after struggling for a while, you aren’t successful, feel free to get in touch with me.)

Beta Binomial Distribution

**Generating the clustered data**

In these examples, I am using 500 clusters, each with cluster size of 40 individuals. There is a cluster-level covariate x that takes on integer values between 1 and 3. The beta distribution is typically defined using two shape parameters usually referenced as α\alphaα and β,\beta,β, where E(Y)=α/(α+β),E(Y) = \alpha / (\alpha + \beta),E(Y)=α/(α+β), and Var(Y)=(αβ)/[(α+β)2(α+β+1)].Var(Y) = (\alpha\beta)/[(\alpha + \beta)^2(\alpha + \beta + 1)].Var(Y)=(αβ)/[(α+β)2(α+β+1)]. In simstudy, the distribution is specified using the mean probability, pm,p\_m,pm​, and a *precision* parameter, ϕβ&gt;0,\phi\_\beta &gt; 0,ϕβ​>0, which is specified using the variance argument. Under this specification, Var(Y)=pm(1−pm)/(1+ϕβ).Var(Y) = p\_m(1 - p\_m)/(1 + \phi\_\beta).Var(Y)=pm​(1−pm​)/(1+ϕβ​). Precision is inversely related to variability: lower precision is higher variability.

In this simple simulation, the cluster probabilities are a function of the cluster-level covariate and precision parameter ϕβ\phi\_\betaϕβ​. Specifically

logodds(pclust)=−2.0+0.65x.logodds(p\_{clust}) = -2.0 + 0.65x.logodds(pclust​)=−2.0+0.65x. The binomial variable of interest bbb is a function of pclustp\_{clust}pclust​ only, and represents a count of individuals in the cluster with a “success”:

library(simstudy)

set.seed(87387)

phi.beta <- 3 # precision

n <- 40 # cluster size

def <- defData(varname = "n", formula = n,

dist = 'nonrandom', id = "cID")

def <- defData(def, varname = "x", formula = "1;3",

dist = 'uniformInt')

def <- defData(def, varname = "p", formula = "-2.0 + 0.65 \* x",

variance = phi.beta, dist = "beta", link = "logit")

def <- defData(def, varname = "b", formula = "p", variance = n,

dist = "binomial")

dc <- genData(500, def)

dc

## cID n x p b

## 1: 1 40 2 0.101696930 4

## 2: 2 40 2 0.713156596 32

## 3: 3 40 1 0.020676443 2

## 4: 4 40 2 0.091444678 4

## 5: 5 40 2 0.139946091 6

## ---

## 496: 496 40 1 0.062513419 4

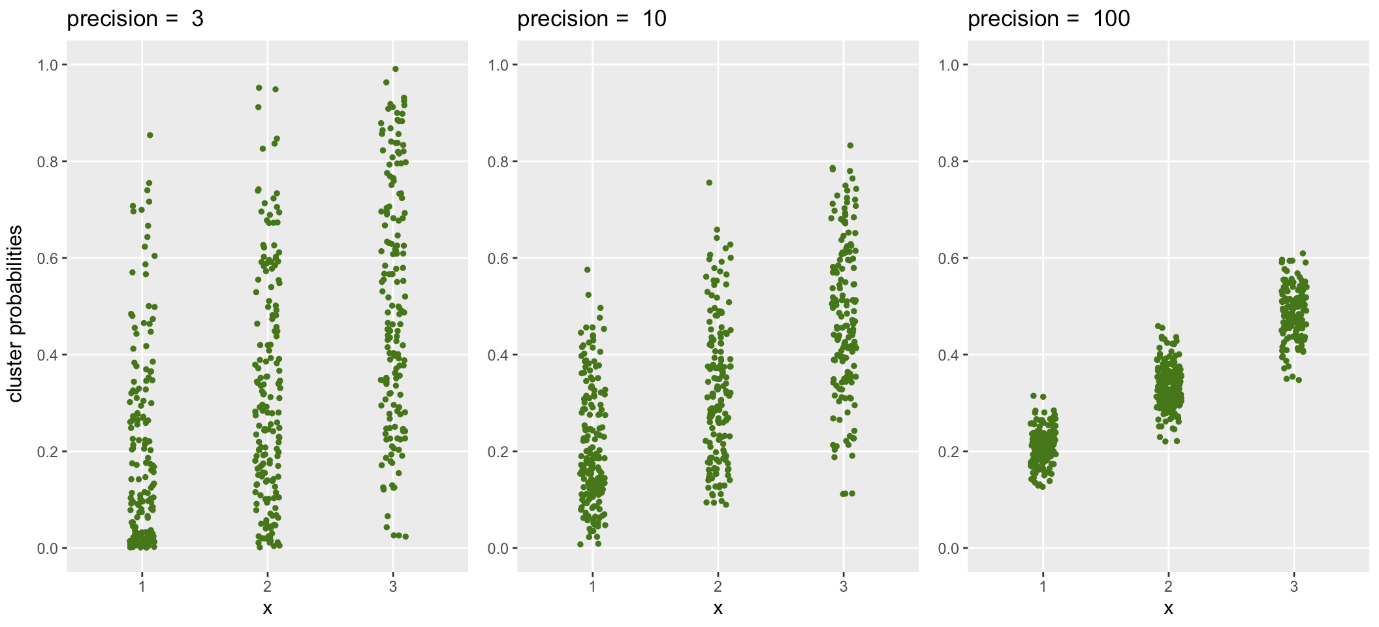
## 497: 497 40 1 0.223149651 5

## 498: 498 40 3 0.452904009 14

## 499: 499 40 2 0.005143594 1

## 500: 500 40 2 0.481283809 16

The generated data with ϕβ=3\phi\_\beta = 3ϕβ​=3 is shown on the left below. Data sets with increasing precision (less variability) are shown to the right:



The relationship of ϕβ\phi\_\betaϕβ​ and variance is made clear by evaluating the variance of the cluster probabilities at each level of xxx and comparing these variance estimates with the theoretical values suggested by parameters specified in the data generation process:

p.clust = 1/(1 + exp(2 - 0.65\*(1:3)))

cbind(dc[, .(obs = round(var(p), 3)), keyby = x],

theory = round( (p.clust\*(1 - p.clust))/(1 + phi.beta), 3))

## x obs theory

## 1: 1 0.041 0.041

## 2: 2 0.054 0.055

## 3: 3 0.061 0.062

**Beta and beta-binomial regression**

Before getting to the GEE estimation, here are two less frequently used regression models: beta and beta-binomial regression. Beta regression may not be super-useful, because we would need to observe (and measure) the probabilities directly. In this case, we randomly generated the probabilities, so it is fair to estimate a regression model to recover the same parameters we used to generate the data! But, back in the real world, we might only observe p^\hat{p}p^​, which results from generating data based on the underlying true ppp. This is where we will need the beta-binomial regression (and later, the GEE model).

First, here is the beta regression using package betareg, which provides quite good estimates of the two coefficients and the precision parameter ϕβ\phi\_\betaϕβ​, which is not so surprising given the large number of clusters in our sample:

library(betareg)

model.beta <- betareg(p ~ x, data = dc, link = "logit")

summary(model.beta)

##

## Call:

## betareg(formula = p ~ x, data = dc, link = "logit")

##

## Standardized weighted residuals 2:

## Min 1Q Median 3Q Max

## -3.7420 -0.6070 0.0306 0.6699 3.4952

##

## Coefficients (mean model with logit link):

## Estimate Std. Error z value Pr(>|z|)

## (Intercept) -2.09663 0.12643 -16.58 <2e-16 \*\*\*

## x 0.70080 0.05646 12.41 <2e-16 \*\*\*

##

## Phi coefficients (precision model with identity link):

## Estimate Std. Error z value Pr(>|z|)

## (phi) 3.0805 0.1795 17.16 <2e-16 \*\*\*

## ---

## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

##

## Type of estimator: ML (maximum likelihood)

## Log-likelihood: 155.2 on 3 Df

## Pseudo R-squared: 0.2388

## Number of iterations: 13 (BFGS) + 1 (Fisher scoring)

The beta-binomial regression model, which is estimated using package aod, is a reasonable model to fit in this case where we have observed binomial outcomes and unobserved underlying probabilities:

library(aod)

model.betabinom <- betabin(cbind(b, n - b) ~ x, ~ 1, data = dc)

model.betabinom

## Beta-binomial model

## -------------------

## betabin(formula = cbind(b, n - b) ~ x, random = ~1, data = dc)

##

## Convergence was obtained after 100 iterations.

##

## Fixed-effect coefficients:

## Estimate Std. Error z value Pr(> |z|)

## (Intercept) -2.103e+00 1.361e-01 -1.546e+01 0e+00

## x 6.897e-01 6.024e-02 1.145e+01 0e+00

##

## Overdispersion coefficients:

## Estimate Std. Error z value Pr(> z)

## phi.(Intercept) 2.412e-01 1.236e-02 1.951e+01 0e+00

##

## Log-likelihood statistics

## Log-lik nbpar df res. Deviance AIC AICc

## -1.711e+03 3 497 1.752e+03 3.428e+03 3.428e+03

A couple of interesting things to note here. First is that the coefficient estimates are pretty similar to the beta regression model. However, the standard errors are slightly higher, as they should be, since we are using only observed probabilities and not the true (albeit randomly selected or generated) probabilities. So, there is another level of uncertainty beyond sampling error.

Second, there is a new parameter: ϕoverdisp\phi\_{overdisp}ϕoverdisp​. What is that, and how does that relate to ϕβ\phi\_\betaϕβ​? The variance of a binomial random variable YYY with a single underlying probability is Var(Y)=np(1−p)Var(Y) = np(1-p)Var(Y)=np(1−p). However, when the underlying probability varies across different subgroups (or clusters), the variance is augmented by ϕoverdisp\phi\_{overdisp}ϕoverdisp​: Var(Y)=np(1−p)[1+(n−1)ϕoverdisp]Var(Y) = np(1-p)[1 + (n-1)\phi\_{overdisp}]Var(Y)=np(1−p)[1+(n−1)ϕoverdisp​]. It turns out to be the case that ϕoverdisp=1/(1+ϕβ)\phi\_{overdisp} = 1/(1+\phi\_\beta)ϕoverdisp​=1/(1+ϕβ​):

round(model.betabinom@random.param, 3) # from the beta - binomial model

## phi.(Intercept)

## 0.241

round(1/(1 + coef(model.beta)["(phi)"]), 3) # from the beta model

## (phi)

## 0.245

The observed variances of the binomial outcome bbb at each level of xxx come quite close to the theoretical variances based on ϕβ\phi\_\betaϕβ​:

phi.overdisp <- 1/(1+phi.beta)

cbind(dc[, .(obs = round(var(b),1)), keyby = x],

theory = round( n\*p.clust\*(1-p.clust)\*(1 + (n-1)\*phi.overdisp), 1))

## x obs theory

## 1: 1 69.6 70.3

## 2: 2 90.4 95.3

## 3: 3 105.2 107.4

**GEE and individual level data**

With individual level binary outcomes (as opposed to count data we were working with before), GEE models are appropriate. The code below generates individual-level for each cluster level:

defI <- defDataAdd(varname = "y", formula = "p", dist = "binary")

di <- genCluster(dc, "cID", numIndsVar = "n", level1ID = "id")

di <- addColumns(defI, di)

di

## cID n x p b id y

## 1: 1 40 2 0.1016969 4 1 0

## 2: 1 40 2 0.1016969 4 2 0

## 3: 1 40 2 0.1016969 4 3 0

## 4: 1 40 2 0.1016969 4 4 0

## 5: 1 40 2 0.1016969 4 5 0

## ---

## 19996: 500 40 2 0.4812838 16 19996 1

## 19997: 500 40 2 0.4812838 16 19997 0

## 19998: 500 40 2 0.4812838 16 19998 1

## 19999: 500 40 2 0.4812838 16 19999 1

## 20000: 500 40 2 0.4812838 16 20000 0

The GEE model provides estimates of the coefficients as well as the working correlation. If we assume an “exchangeable” correlation matrix, in which each individual is correlated with all other individuals in the cluster but is not correlated with individuals in other clusters, we will get a single correlation estimate, which is labeled as *alpha* in the GEE output:

library(geepack)

geefit <- geeglm(y ~ x, family = "binomial", data = di,

id = cID, corstr = "exchangeable" )

summary(geefit)

##

## Call:

## geeglm(formula = y ~ x, family = "binomial", data = di, id = cID,

## corstr = "exchangeable")

##

## Coefficients:

## Estimate Std.err Wald Pr(>|W|)

## (Intercept) -2.05407 0.14854 191.2 <2e-16 \*\*\*

## x 0.68965 0.06496 112.7 <2e-16 \*\*\*

## ---

## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

##

## Correlation structure = exchangeable

## Estimated Scale Parameters:

##

## Estimate Std.err

## (Intercept) 1.001 0.03192

## Link = identity

##

## Estimated Correlation Parameters:

## Estimate Std.err

## alpha 0.2493 0.01747

## Number of clusters: 500 Maximum cluster size: 40

In this case, *alpha* (α\alphaα) is estimated at 0.25, which is quite close to the previous estimate of ϕoverdisp\phi\_{overdisp}ϕoverdisp​, 0.24. So, it appears to be the case that if we have a target correlation α\alphaα, we know the corresponding ϕβ\phi\_\betaϕβ​ to use in the beta-binomial data generation process. That is, ϕβ=(1−α)/α\phi\_\beta = (1 - \alpha)/\alphaϕβ​=(1−α)/α.

While this is certainly not a proof of anything, let’s give it a go with a target α=0.44\alpha = 0.44α=0.44:

phi.beta.new <- (1-0.44)/0.44

def <- updateDef(def, "p", newvariance = phi.beta.new)

dc2 <- genData(500, def)

di2 <- genCluster(dc2, "cID", numIndsVar = "n", level1ID = "id")

di2 <- addColumns(defI, di2)

geefit <- geeglm(y ~ x, family = "binomial", data = di2,

id = cID, corstr = "exchangeable" )

summary(geefit)

##

## Call:

## geeglm(formula = y ~ x, family = "binomial", data = di2, id = cID,

## corstr = "exchangeable")

##

## Coefficients:

## Estimate Std.err Wald Pr(>|W|)

## (Intercept) -2.0472 0.1844 123.2 <2e-16 \*\*\*

## x 0.6994 0.0804 75.6 <2e-16 \*\*\*

## ---

## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

##

## Correlation structure = exchangeable

## Estimated Scale Parameters:

##

## Estimate Std.err

## (Intercept) 1 0.0417

## Link = identity

##

## Estimated Correlation Parameters:

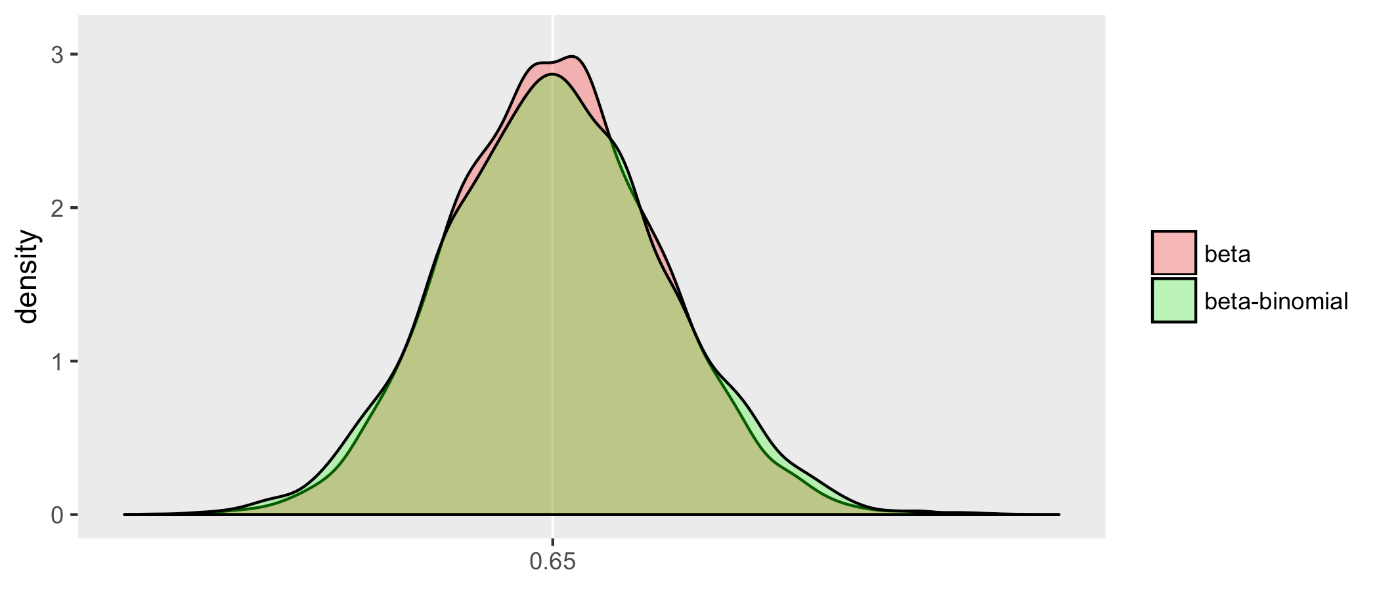
## Estimate Std.err

## alpha 0.44 0.0276

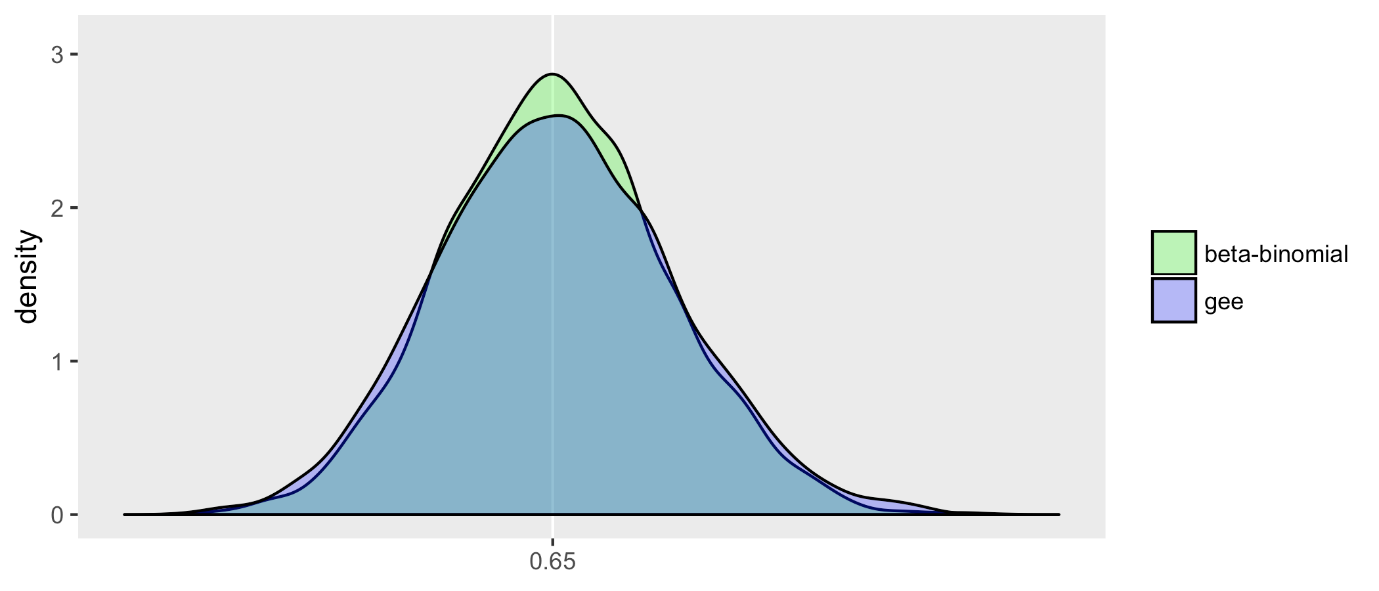
## Number of clusters: 500 Maximum cluster size: 40

**Addendum**

Above, I suggested that the estimator of the effect of x based on the beta model will have less variation than the estimator based on the beta-binomial model. I drew 5000 samples from the data generating process and estimated the models each time. Below is a density distribution of the estimates of each of the models from all 5000 iterations. As expected, the beta-binomial process has more variability, as do the related estimates; we can see this in the relative “peakedness”" of the beta density:



Also based on these 5000 iterations, the GEE model estimation appears to be less efficient than the beta-binomial model. This is not surprising since the beta-binomial model was the actual process that generated the data (so it is truly the correct model). The GEE model is robust to mis-specification of the correlation structure, but the price we pay for that robustness is a slightly less precise estimate (even if we happen to get the correlation structure right):



Binomial & Poisson Distribution

In many situations, when count data are modeled, it turns out that the variance of the data exceeds the mean (a situation called over-dispersion). In this case an alternative model is used that allows for the greater variance, which is based on the negative binomial distribution. It turns out that if the negative binomial distribution has mean μ\muμ, it has a variance of μ+θμ2\mu + \theta \mu^2μ+θμ2, where θ\thetaθ is called a dispersion parameter. If θ=0\theta = 0θ=0, we have the Poisson distribution, but otherwise the variance of a negative binomial random variable will exceed the variance of a Poisson random variable as long as they share the same mean, because μ&gt;0\mu &gt; 0μ>0 and θ≥0\theta \ge 0θ≥0.

We can see this by generating data from each distribution with mean 15, and a dispersion parameter of 0.2 for the negative binomial. We expect a variance around 15 for the Poisson distribution, and 60 for the negative binomial distribution.

library(simstudy)

library(ggplot2)

# for a less cluttered look

theme\_no\_minor <- function(color = "grey90") {

theme(panel.grid.minor = element\_blank(),

panel.background = element\_rect(fill="grey95")

)

}

options(digits = 2)

# define data

defC <- defCondition(condition = "dist == 0", formula = 15,

dist = "poisson", link = "identity")

defC <- defCondition(defC, condition = "dist == 1", formula = 15,

variance = 0.2, dist = "negBinomial",

link = "identity")

# generate data

set.seed(50)

dt <- genData(500)

dt <- trtAssign(dt, 2, grpName = "dist")

dt <- addCondition(defC, dt, "y")

genFactor(dt, "dist", c("Poisson", "Negative binomial"))

# compare distributions

dt[, .(mean = mean(y), var = var(y)), keyby = fdist]

## fdist mean var

## 1: Poisson 15 15

## 2: Negative binomial 15 54

ggplot(data = dt, aes(x = y, group = fdist)) +

geom\_density(aes(fill=fdist), alpha = .4) +

scale\_fill\_manual(values = c("#808000", "#000080")) +

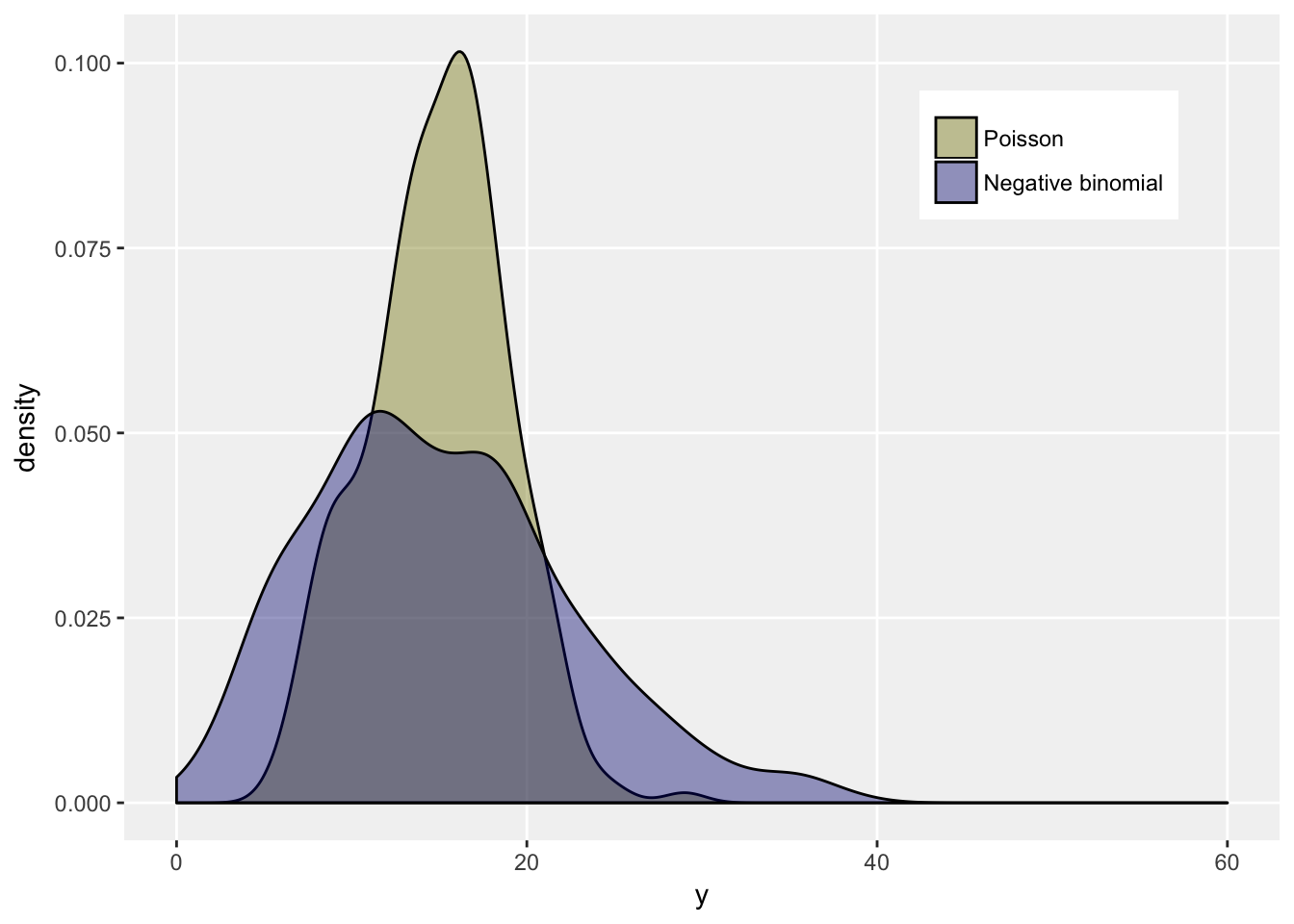
scale\_x\_continuous(limits = c(0,60),

breaks = seq(0, 60, by = 20)) +

theme\_no\_minor() +

theme(legend.title = element\_blank(),

legend.position = c(0.80, 0.83))



### Underestimating standard errors

In the context of a regression, misspecifying a model as Poisson rather than negative binomial, can lead to an underestimation of standard errors, even though the point estimates may be quite reasonable (or may not). The Poisson model will force the variance estimate to be equal to the mean at any particular point on the regression curve. The Poisson model will effectively ignore the true extent of the variation, which can lead to problems of interpretation. We might conclude that there is an association when in fact there is none.

In this simple simulation, we generate two predictors (xxx and bbb) and an outcome (yyy). The outcome is a function of xxx only:

library(broom)

library(MASS)

# Generating data from negative binomial dist

def <- defData(varname = "x", formula = 0, variance = 1,

dist = "normal")

def <- defData(def, varname = "b", formula = 0, variance = 1,

dist = "normal")

def <- defData(def, varname = "y", formula = "0.9 + 0.6\*x",

variance = 0.3, dist = "negBinomial", link = "log")

set.seed(35)

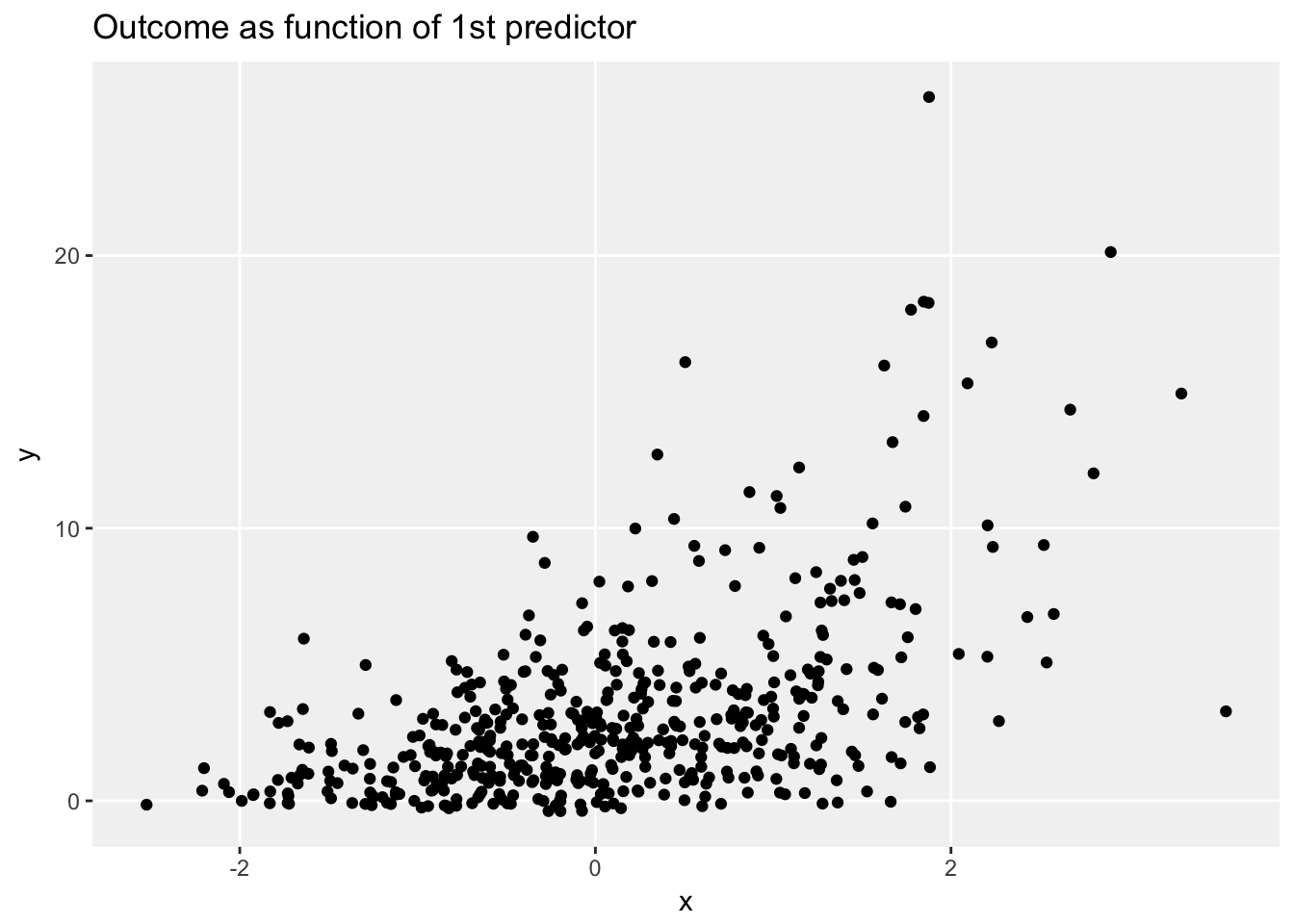
dt <- genData(500, def)

ggplot(data = dt, aes(x=x, y = y)) +

geom\_jitter(width = .1) +

ggtitle("Outcome as function of 1st predictor") +

theme\_no\_minor()

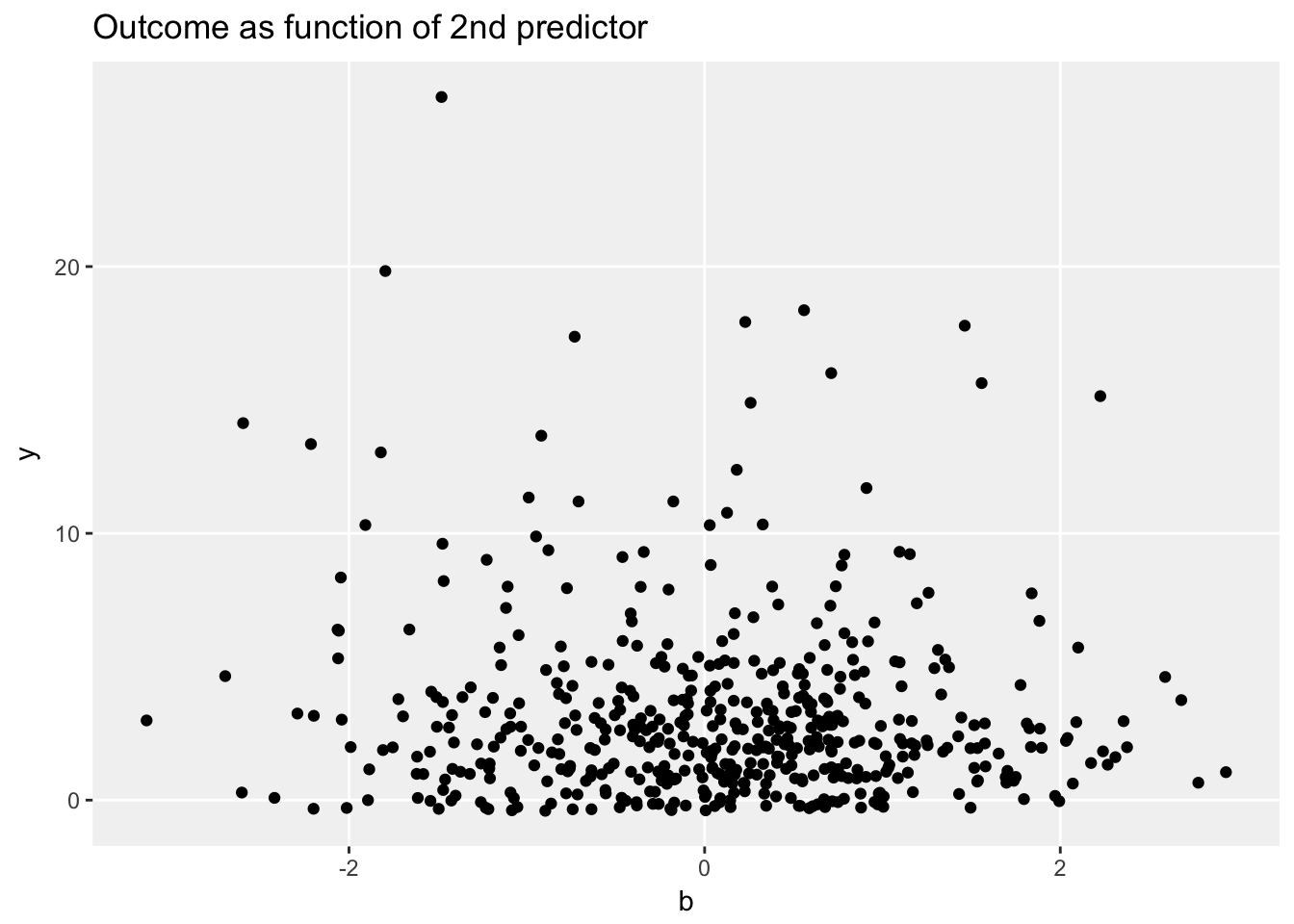


ggplot(data = dt, aes(x=b, y = y)) +

geom\_jitter(width = 0) +

ggtitle("Outcome as function of 2nd predictor") +

theme\_no\_minor()



I fit two models using both predictors. The first assumes (incorrectly) a Poisson distribution, and the second assumes (correctly) a negative binomial distribution. We can see that although the point estimates are quite close, the standard error estimates for the predictors in the Poisson model are considerably greater (about 50% higher) than the negative binomial model. And if we were basing any conclusion on the p-value (which is not always the obvious way to do [things](http://www.stat.columbia.edu/~gelman/research/unpublished/abandon.pdf)), we might make the wrong call since the p-value for the slope of bbb is estimated to be 0.029. Under the correct model model, the p-value is 0.29.

glmfit <- glm(y ~ x + b, data = dt, family = poisson (link = "log") )

tidy(glmfit)

## term estimate std.error statistic p.value

## 1 (Intercept) 0.956 0.030 32.3 1.1e-228

## 2 x 0.516 0.024 21.9 1.9e-106

## 3 b -0.052 0.024 -2.2 2.9e-02

nbfit <- glm.nb(y ~ x + b, data = dt)

tidy(nbfit)

## term estimate std.error statistic p.value

## 1 (Intercept) 0.954 0.039 24.2 1.1e-129

## 2 x 0.519 0.037 14.2 7.9e-46

## 3 b -0.037 0.036 -1.1 2.9e-01

A plot of the fitted regression curve and confidence bands of bbb estimated by each model reinforces the difference. The lighter shaded region is the wider confidence band of the negative binomial model, and the darker shaded region the based on the Poisson model.

newb <- data.table(b=seq(-3,3,length = 100), x = 0)

poispred <- predict(glmfit, newdata = newb, se.fit = TRUE,

type = "response")

nbpred <-predict(nbfit, newdata = newb, se.fit = TRUE,

type = "response")

poisdf <- data.table(b = newb$b, y = poispred$fit,

lwr = poispred$fit - 1.96\*poispred$se.fit,

upr = poispred$fit + 1.96\*poispred$se.fit)

nbdf <- data.table(b = newb$b, y = nbpred$fit,

lwr = nbpred$fit - 1.96\*nbpred$se.fit,

upr = nbpred$fit + 1.96\*nbpred$se.fit)

ggplot(data = poisdf, aes(x=b, y = y)) +

geom\_line() +

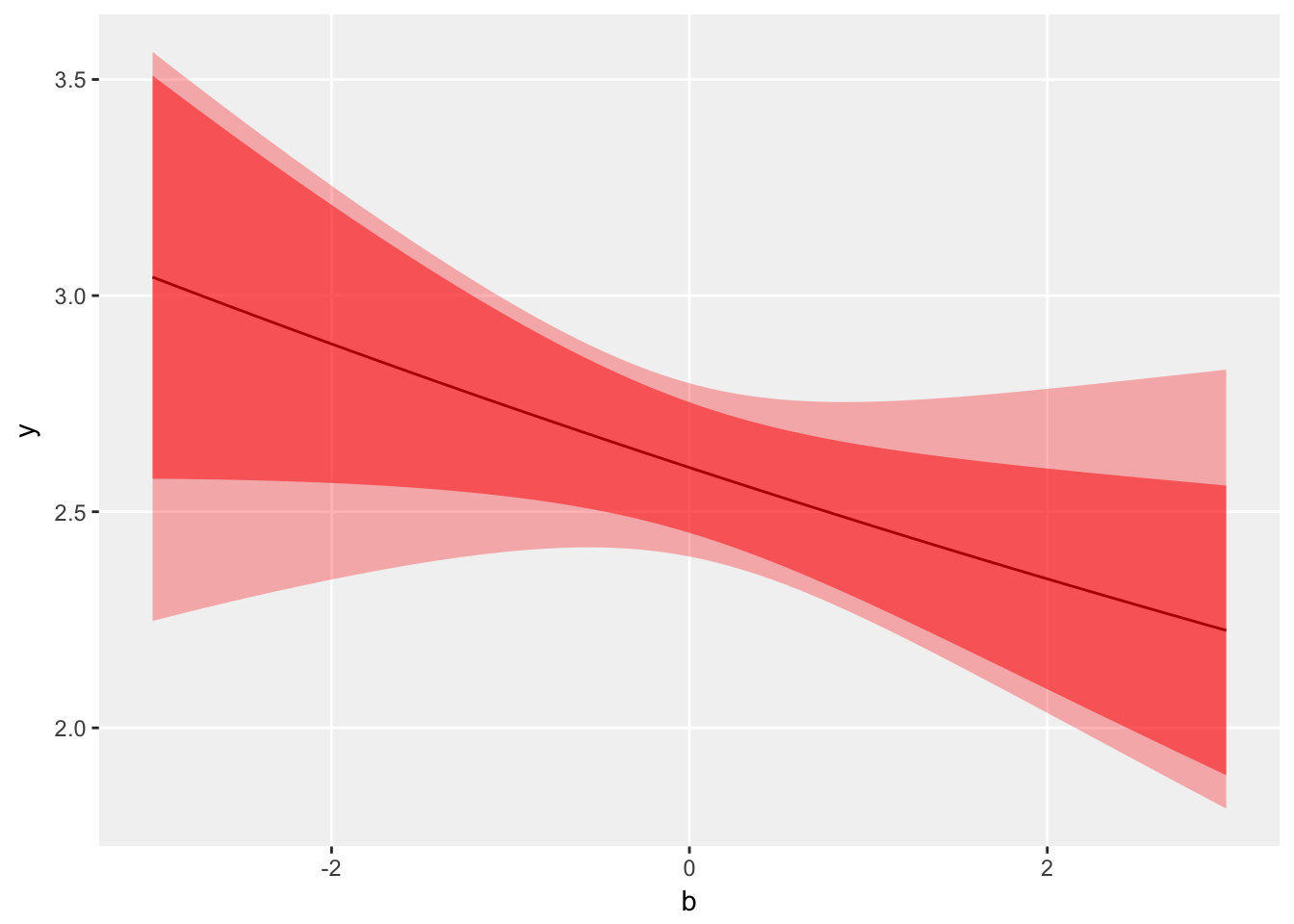
geom\_ribbon(data=nbdf, aes(ymin = lwr, ymax=upr), alpha = .3,

fill = "red") +

geom\_ribbon(aes(ymin = lwr, ymax=upr), alpha = .5,

fill = "red") +

theme\_no\_minor()



And finally, if we take 500 samples of size 500, and estimate slope for bbb each time and calculate the standard deviation of those estimates, it is quite close to the standard error estimate we saw in the model of the original simulated data set using the negative binomial assumption (0.036). And the mean of those estimates is quite close to zero, the true value.

result <- data.table()

for (i in 1:500) {

dt <- genData(500, def)

glmfit <- glm(y ~ x + b, data = dt, family = poisson)

nbfit <- glm.nb(y ~ x + b, data = dt)

result <- rbind(result, data.table(bPois = coef(glmfit)["b"],

bNB = coef(nbfit)["b"])

)

}

result[,.(sd(bPois), sd(bNB))] # observed standard error

## V1 V2

## 1: 0.037 0.036

result[,.(mean(bPois), mean(bNB))] # observed mean

## V1 V2

## 1: 0.0025 0.0033

### Negative binomial as mixture of Poissons

An interesting relationship between the two distributions is that a negative binomial distribution can be generated from a mixture of individuals whose outcomes come from a Poisson distribution, but each individual has her own rate or mean. Furthermore, those rates must have a specific distribution - a Gamma. (For much more on this, you can take a look [here](https://probabilityandstats.wordpress.com/tag/poisson-gamma-mixture/).) Here is a little simulation:

mu = 15

disp = 0.2

# Gamma distributed means

def <- defData(varname = "gmu", formula = mu, variance = disp,

dist = "gamma")

# generate data from each distribution

defC <- defCondition(condition = "nb == 0", formula = "gmu",

dist = "poisson")

defC <- defCondition(defC, condition = "nb == 1", formula = mu,

variance = disp, dist = "negBinomial")

dt <- genData(5000, def)

dt <- trtAssign(dt, 2, grpName = "nb")

genFactor(dt, "nb", labels = c("Poisson-Gamma", "Negative binomial"))

dt <- addCondition(defC, dt, "y")

# means and variances should be very close

dt[, .(Mean = mean(y), Var = var(y)), keyby = fnb]

## fnb Mean Var

## 1: Poisson-Gamma 15 62

## 2: Negative binomial 15 57

# plot

ggplot(data = dt, aes(x = y, group = fnb)) +

geom\_density(aes(fill=fnb), alpha = .4) +

scale\_fill\_manual(values = c("#808000", "#000080")) +

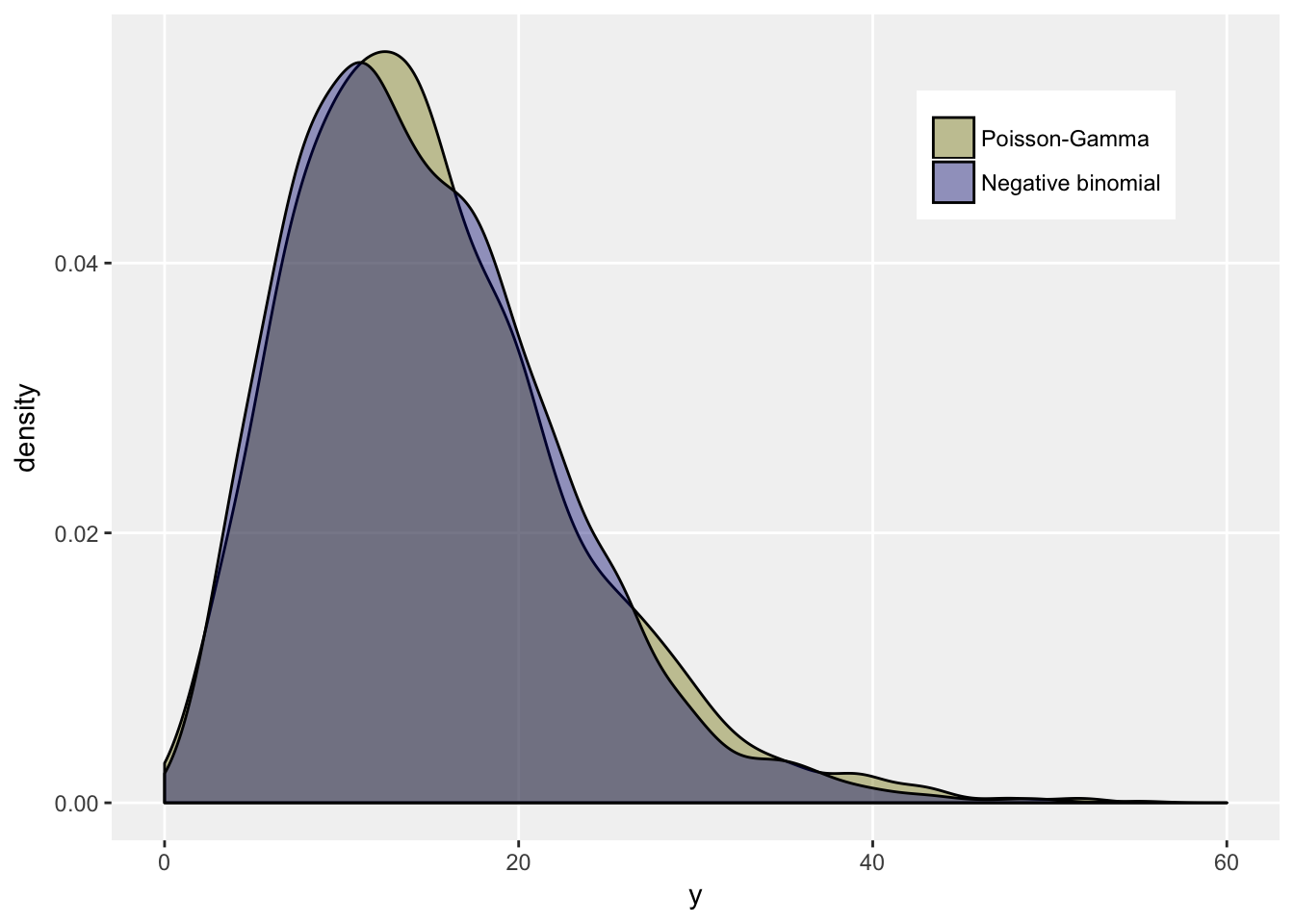
scale\_x\_continuous(limits = c(0,60),

breaks = seq(0, 60, by = 20)) +

theme\_no\_minor() +

theme(legend.title = element\_blank(),

legend.position = c(0.80, 0.83))



```