Stat 5421 Lecture Notes: Completion of Exponential Families

Charles J. Geyer

July 27, 2023

Contents

1 License

2 R

```
library(alabama)
```

```
## Loading required package: numDeriv
```

```
library(rcdd)
```

```
## If you want correct answers, use rational arithmetic.
## See the Warnings sections in help pages for
##     functions that do computational geometry.
```

```
library(llmdr, lib.loc = "~/Software/llmdr/package/llmdr.Rcheck")
library(sfsmisc)
library(MASS)
library(glmbb)
library(nnet)
library(CatDataAnalysis)
```

- The version of R used to make this document is 4.3.1.

- The version of the `alabama` package used to make this document is 2022.4.1.
- The version of the `numDeriv` package used to make this document is 2016.8.1.1.
- The version of the `rcdd` package used to make this document is 1.5.2.
- The version of the `llmdr` package used to make this document is 0.1.
- The version of the `sfsmisc` package used to make this document is 1.1.15.
- The version of the `MASS` package used to make this document is 7.3.60.
- The version of the `glmbb` package used to make this document is 0.5.1.
- The version of the `nnet` package used to make this document is 7.3.19.
- The version of the `CatDataAnalysis` package used to make this document is 0.1.5.
- The version of the `rmarkdown` package used to make this document is 2.23.

At present, R package `llmdr` is not publicly available. That is why it is loaded from a nonstandard location on the author's computer. It will be available soon.

3 History of Theory

Maximum likelihood estimates (MLE) do not have to exist. Theoretically, the problem of when MLE exist in regular full exponential families of distributions has been completely understood for a long time.

For exponential families having finite support, this theory is due to Barndorff-Nielsen (1978), Theorem 9.15 and the surrounding discussion. Finite support means there are only a finite number of possible values of the canonical statistic vector, as happens in logistic regression and log-linear models for categorical data when multinomial or product-multinomial sampling is assumed.

Brown (1986), Chapter 6, Section "Aggregate Exponential Families" extended Barndorff-Nielsen's theory to exponential families with countable support under certain regularity conditions. Countable support means we have a discrete probability distribution, and Poisson regression with log link and log-linear models for categoriccal data when Poisson sampling is assumed satisfy Brown's regularity conditions.

Geyer (2009) reworked this theory to use the concepts

- direction of recession (DOR)
- direction of constancy (DOC)
- generic direction of recession (GDOR)

4 Directions in the Canonical Parameter Space

Following Geyer (2009) we are going to be sloppy about the mathematical meaning of the word *direction*. Strictly speaking, it is the direction in which a vector points, so every positive scalar multiple of a vector points in the same direction. So a direction is not a vector. Vectors have length, directions do not. Also the zero vector does not point in any direction. But we are just going to call vectors directions, and we are going to allow the zero vector. There is a reason for this in the correspondence between DOR and normal vectors (Geyer (2009), Theorem 3), which we do not want to discuss. Mostly this does not cause confusion. Occasionally it does lead to some clumsy language.

When the MLE does not exist in the original model (OM), this is because there is a DOR that is not a DOC, (Geyer (2009), Theorems 1, 3, and 4).

Here are the definitions of these concepts that give them their names.

- A vector δ in the canonical parameter space of a regular full exponential family is a DOR if the log likelihood function l is nondecreasing in that direction, meaning for any θ in the canonical parameter space and any positive real number s we have $l(\theta) \le l(\theta + s\delta)$.
- A vector δ in the canonical parameter space of a regular full exponential family is a DOC if the log likelihood function l is constant in that direction, meaning for any any θ in the canonical parameter space and any real number s we have $l(\theta) = l(\theta + s\delta)$.

Every DOC is a DOR, but not vice versa.

And here are some equivalent characterizations. In them Y denotes the canonical statistic vector, thought of as a random vector having some probability distribution in the family, and y denotes its observed value.

- A vector δ in the canonical parameter space of a regular full exponential family is a DOC if and only if for any real number s and any θ in the canonical parameter space θ and $\theta + s\delta$ correspond to the same distribution.
- A vector δ in the canonical parameter space of a regular full exponential family is a DOC if and only if $\langle Y, \delta \rangle$ is a constant random variable.
- A vector δ in the canonical parameter space of a regular full exponential family is a DOR if and only $\langle Y, \delta \rangle \le \langle y, \delta \rangle$ with probability one.

In summary,

- δ is a DOC if and only if $\langle Y, \delta \rangle$ is constant,
- δ is a DOR if and only if $\langle y, \delta \rangle$ is extreme, the upper endpoint of the range of $\langle Y, \delta \rangle$, and
- δ is a DOC if and only if and only if θ and $\theta + \delta$ correspond to the same distribution.

The last bullet point here says that DOC are about identifiability, If there is a DOC, then we need to drop some parameters to gain identifiability. Usually, the software can do this for us.

The other two bullet points here are about maximum likelihood.

The MLE exists in the OM if and only if every DOR is a DOC (Geyer (2009), Theorem 4).

What happens when the MLE does not exist in the OM is the subject of the following section.

5 Limits

If we take straight line limits in the direction of a DOR δ then all of the probability goes to as far as it can go in that direction (Geyer (2009), Theorem 6, but similar theorems are found in Barndorff-Nielsen (1978), Brown (1986), and Geyer (1990)). Probability piles up on the subset of the support

$$H = \{\, Y : \langle Y - y, \delta \rangle = 0 \,\} \tag{1}$$

The probability distribution for $\theta + s\delta$ converges as $s \to \infty$ to the conditional probability distribution for θ of Y given $Y \in H$. This family of conditional distributions (for all θ) is an exponential family of distributions that Geyer (2009) calls a *limiting conditional model* (LCM).

It is wrong to think of distributions in the LCM only as conditional distributions or only as limit distributions. They are both.

Since the log likelihood is nondecreasing along a DOR, the log likelihood for an LCM lies on or above the log likelihood for the OM. Hence maximizing the likelihood in the LCM maximizes the likelihood in the OM too.

The canonical parameters do not converge as we take these limits. They go off to infinity.

But the probability distributions converge. So the MLE *distribution* in the LCM (if one exists, more on this presently) can be thought of as the MLE.

Moreover, not only distributions converge. Under the regularity conditions of Brown, which hold for all statistical models studied in this course, moments of all orders converge (Eck and Geyer (2021), Theorem 7), for example, the mean of the distribution in the OM for parameter $\theta + s\delta$ converges to the mean of the distribution for the parameter θ in the LCM as $s \to \infty$. And the same goes for the variance and other moments.

Thus *mean value parameters* are well behaved. When we take limits to complete exponential families, means converge.

Every LCM is itself an exponential family, just a different exponential family from the OM. The LCM and the OM have the same canonical statistic and the same canonical parameter. The LCM just restricts the canonical statistic to the event H described above.

If δ is a GDOR of the OM, then δ is a DOC of the LCM. This is clear from the fact that the LCM conditions on the event $Y \in H$ with H given by (1). This is the reason we have to allow DOC. Otherwise, we cannot talk about the OM and LCM using the same parameters.

Since the LCM is an exponential family, we can tell when the MLE exists in it. Just apply the theory described above to it. The MLE will exist in the LCM if and only if every DOR of the LCM (not OM) is a DOC of the LCM (not OM). And (Geyer (2009), Section 3.6) show this happens whenever the DOR of the OM that we used to form the LCM was *generic*, not in any way special. This we call a *generic direction of recession* (GDOR).

A more precise characterization of GDOR will be given at the end of Section 8 below.

When we find a GDOR and its corresponding LCM, then the MLE in the LCM is the MLE *distribution* and the mean value of this distribution is the MLE *mean value*. And this MLE satisfies the observed-equals-expected property, because it is the MLE for an exponential family (the LCM). Hence the observed-equals-expected property always holds. Similarly, the sufficient dimension reduction property and maximum entropy properties always hold. It does not matter whether the MLE is found in the OM or an LCM.

One might ask when the MLE is in an LCM, what is the statistical model? The model cannot be the OM, because if that is so, then there is no MLE. We have to say that the model is the OM and all of its limits. Geyer (2009) calls that model the *Barndorff-Nielsen completion* of the OM.

The Barndorff-Nielsen completion has no canonical parameterization, because when you take limits canonical parameters do not converge (they go off to infinity). But it is a family of distributions (the OM and all of its LCM). Also (under the conditions of Brown) it has a mean value parameterization. Every distribution in the Barndorff-Nielsen completion has a mean vector, and mean vectors do converge as limits are taken to get LCM.

Each LCM is an exponential family that has a canonical parameterization, but the Barndorff-Nielsen completion as a whole does not have a canonical parameterization. The canonical parameterizations of the LCM do not fit together to make one parameterization.

6 History of Computing

The thesis of your humble author (Geyer (1990), Chapter 2), included an algorithm using repeated linear programming to discover DOR that are not DOC for exponential families. With software available at the time it was very clumsy but was actually used to find MLE in LCM for some very complicated statistical models (Geyer and Thompson, 1992).

Geyer (2009) used various functions in R package `rcdd` (Geyer *et al.*, 2021) to find GDOR. This software was also somewhat clumsy to use (there was no model fitting function comparable to R function `glm`) but only a dozen or so lines of code were enough to find out whether the MLE existed in the OM or otherwise a GDOR and its LCM and the MLE in that LCM. But this code was slow on big problems. Eck and Geyer (2021) have an example (dataset `bigcategorical` in R package `llmdr`), a contingency table with 1024 cells

and 781 parameters, that took a little over 3 days and 4 hours of computing time to do with the methods of this paragraph.

Eck and Geyer (2021) used R function `glm` to maximize the likelihood getting canonical parameter estimates close to infinity by using small tolerances and then looking for directions in which the Fisher information matrix is nearly singular (approximate null eigenvectors). This algorithm was fast but has problems with inexactness of computer arithmetic. So it is difficult to know exactly when an approximate null eigenvector of the Fisher information matrix is close to being a GDOR.

R package `llmdr` (Geyer, 2022) uses repeated linear programming but different linear programs from those used by Geyer (1990) and Geyer (2009). It also uses the fast linear programming (R package `glpkAPI`, Gelius-Dietrich (2021)) that scales to large problems (it uses sparse matrix arithmetic). But `llmdr` can also use infinite-precision rational arithmetic from R packages `rcdd` (Geyer *et al.*, 2021) and `gmp` (Lucas *et al.*, 2021) to prove, if requested, that its results are correct. The big categorical example of Eck and Geyer (2021), mentioned above, that took over 3 days to do with the methods of Geyer (2009) took only 150 milliseconds to do with R function `llmdr` in R package `llmdr`. If proofs were requested, they took, 25 minutes (slow, but still an improvement over the methods of Geyer (2009)).

So we finally have, in this fourth generation algorithm, methods that are fast, easy, and safe enough for ordinary users to use.

7 The Binomial Family of Distributions

The binomial family of distributions as usually taught has usual parameter π and usual parameter space $0 \le \pi \le 1$.

The binomial family of distributions thought of as an exponential family has canonical parameter $\theta = \operatorname{logit}(\pi)$ and has canonical parameter space $-\infty < \theta < +\infty$ and this corresponds to usual parameter space $0 < \pi < 1$.

We have lost the distributions for $\pi = 0$ and $\pi = 1$. The reason why these distributions cannot be in the exponential family is all distributions in an exponential family have the same support. There is also the problem that $\operatorname{logit}(0)$ and $\operatorname{logit}(1)$ are undefined, so there are no canonical parameter values for those distributions to have (canonical parameters have to be real numbers or real vectors, infinite values are not allowed).

This is a one-parameter family of distributions so its canonical parameter space is a one-dimensional vector space. In a one-dimensional vector space, there are only two directions: toward $+\infty$ and toward $-\infty$. There are an infinite number of vectors, but only two directions those vectors can point in.

The log likelihood for the binomial distribution is

$$l(\pi) = y \log(\pi) + (n - y) \log(1 - \pi)$$

When $y = 0$, this becomes

$$l(\pi) = n \log(1 - \pi)$$

and this is a strictly increasing function of π and θ in the minus direction (toward $-\infty$). Because logit is a monotone function, as θ decreases, π also decreases, $1 - \pi$ increases, and $\log(1 - \pi)$ increases. So -1 (or any negative number, thought of as a vector in the one-dimensional vector space $\mathbb{R}$) is a DOR.

And what happens to the probabilities as we take limits? There are two cases. For $y = 0$

$$f(y) = (1 - \pi)^n \to 1, \qquad \text{as } \pi \to 0 \text{ and } \theta \to -\infty.$$

And for $y > 0$

$$f(y) = \pi^y (1 - \pi)^{n-y} \to 0, \qquad \text{as } \pi \to 0 \text{ and } \theta \to -\infty.$$

Thus binomial distributions with usual parameter π and canonical parameter θ converge to the completely degenerate distribution concentrated at zero as $\pi \to 0$ and $\theta \to -\infty$.

Now the theory described above says the MLE does not exist in the OM (because -1 is a DOR that is not a DOC) and the LCM is the distribution concentrated at $Y = 0$. This is the only distribution in the LCM, so the LCM has no identifiable parameters, and this only distribution is the MLE distribution. This distribution has mean zero and variance zero, which are the limits of the mean π and variance $\pi(1 - \pi)$ as $\pi \to 0$ and $\theta \to -\infty$.

This seems like a crazy amount of theory to throw at such a simple problem. And it is overkill for a one-dimensional problem like this. But, as we shall soon, see. Even two dimensional problems can be hard for humans to figure out.

So we need the computer to use the theory above to find the answers.

By the way, a similar story could be told about when the data are $y = n$.

8 Canonical Affine Submodels

Canonical affine submodels of regular full exponential families are again regular full exponential families. If a is the offset vector and M is the model matrix, then

$$\theta = a + M\beta$$

is the relationship between

- the saturated model canonical parameter vector θ and
- the canonical affine submodel canonical parameter vector β.

And

$$\tau = M^T \mu$$

is the relationship between

- the saturated model mean value parameter vector μ and
- the canonical affine submodel mean value parameter vector τ.

And

- the saturated model canonical statistic vector is y and
- the canonical affine submodel canonical statistic vector is $M^T y$.

So the only difference between saturated models and canonical affine submodels is that we replace y with $M^T y$.

- A vector δ in the canonical parameter space of a canonical affine submodel a DOC if and only if $\langle M^T(Y - y), \delta \rangle = \langle Y - y, M\delta \rangle$ is zero with probability one.
- A vector δ in the canonical parameter space of a canonical affine submodel is a DOR if and only if for any $\langle M^T(Y - y), \delta \rangle = \langle Y - y, M\delta \rangle$ is nonpositive with probability one.

A vector δ is a DOR for the submodel if and only if $\eta = M\delta$ is a DOR for the saturated model. Thus

- for Poisson sampling η is a DOR if and only if $\eta_i \le 0$ for all i (because the Poisson distribution has no upper bound) and $\eta_i < 0$ implies $y_i = 0$ (because zero is the lower bound),
- for binomial sampling η is a DOR if and only if $\eta_i < 0$ implies $y_i = 0$ (because zero is the lower bound) and $\eta_i > 0$ implies $y_i = n_i$ (because n_i is the lower bound).

Multinomial and product multinomial sampling is considerably more complicated so we defer discussion of that.

But already we can see why this problem is hard. We have to find a δ such that $\eta = M\delta$ satisfies the above conditions. And for complicated models that is complicated. Fortunately, the computer can do it for us.

When we have found a DOR η, the LCM says

- if $\eta_i \neq 0$, then the LCM conditions on $Y_i = y_i$ and
- if $\eta_i = 0$, then the LCM makes no restrictions on Y_i.

Hence the LCM is found by fitting the model to the cases with $\eta_i = 0$. When we lose cases like this, we may also have more non-identifiable parameters, but the computer can figure that out.

Now we can be a bit more precise about what a GDOR is. For Poisson and binomial sampling, a vector δ is a DOR has the the largest possible number of nonzero components so the LCM conditions on the largest possible number of number of components of the response vector. Of course, you have no way to guess what subset of components is "largest possible" but, again, the computer can figure it out.

9 Separation versus Degeneracy

Agresti (2013), Section 6.5.1, uses the terminology of *complete separation* and *quasi-complete separation* to discuss the issues in this area.

We prefer the terminology of *complete degeneracy* and *partial degeneracy* to describe these issues.

Agresti's terms only apply to logistic regression. Our terms only apply to all exponential families.

Complete separation occurs if all cases are plotted in regressor space, the successes and failures are labeled with different colors, and there is a hyperplane in predictor space that separates them, all of one color on one side and all of the other color on the other side. An example of such a plot is Figure 7 below.

This corresponds to an LCM that is *completely degenerate*, concentrated at a single point. The MLE distribution says the only possible value of the response vector was the observed value.

Quasi-complete separation occurs if all cases are plotted in predictor space, the successes and failures are labeled with different colors, and there is a hyperplane in predictor space that weakly separates them, all of one color on one side or in the hyperplane and all of the other color on the other side or in the hyperplane. An example of such a plot is Figure 14 below.

This corresponds to an LCM that is *partially degenerate*, not concentrated at a single point but having support smaller than the OM. The MLE distribution is not completely degenerate but does give probability zero to some outcomes that have positive probability in the OM.

10 Complete Separation Example of Agresti

10.1 Data

Section 6.5.1 of Agresti (2013) introduces the notion of complete separation with the following example.

```
x <- seq(10, 90, 10)
x <- x[x != 50]
y <- as.numeric(x > 50)
data.frame(x, y)
```

```
##    x y
## 1 10 0
## 2 20 0
## 3 30 0
## 4 40 0
## 5 60 1
## 6 70 1
## 7 80 1
## 8 90 1
```

The following figure shows these data.

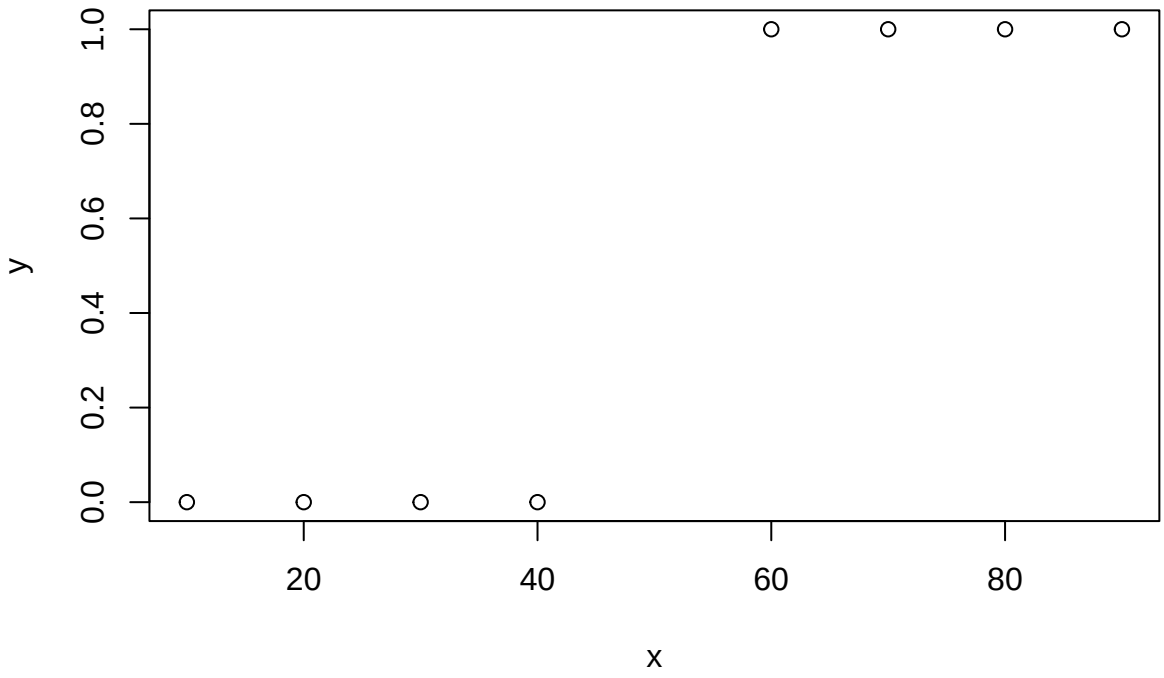


Figure 1: Logistic Regression Data for Complete Separation Example of Agresti

10.2 Attempt to Fit Model

Suppose we want to do "simple" logistic regression (one predictor x plus intercept, so the model is two-dimensional). Let's try to do it naively.

```
gout <- glm(y ~ x, family = binomial)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
summary(gout)
```

```
##
## Call:
## glm(formula = y ~ x, family = binomial)
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -118.158 296046.187       0        1
## x                2.363   5805.939       0        1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1.1090e+01  on 7  degrees of freedom
```

```
## Residual deviance: 2.1827e-10  on 6  degrees of freedom
## AIC: 4
##
## Number of Fisher Scoring iterations: 25
```

R function `glm` does give a warning. But what are you supposed to do about it? If the output of the R function `summary` is to be taken seriously, you cannot tell whether either regression coefficient is nonzero. As we shall see, that is complete nonsense. Both parameters have to go to infinity (one to plus infinity, the other to minus infinity) in order to maximize the likelihood.

In fact, these data are not analyzable by the R function `glm` and its associated functions (generic functions having methods for class `"glm"`).

At least it did give a (not very clear) warning that its results are absolute rubbish. But that is no consolation because

- it doesn't give you any options to do the Right Thing and
- it gives both false negative and false positives on these warnings, so they are nonsense too. You cannot rely on anything R function `glm` does.

10.3 Log-Linear Models Done Right

So we switch to R function `llmdr` which stands for log-linear models done right (a take-off on the name of a well-known linear algebra textbook) in R package `llmdr` (Geyer, 2022).

```
lout <- llmdr(y ~ x, family = "binomial")
summary(lout)
```

```
##
## Call:
## llmdr(formula = y ~ x, family = "binomial")
##
##             Estimate   GDOR Std. Error z value Pr(>|z|)
## (Intercept)       NA -1.250         NA      NA       NA
## x                 NA  0.025         NA      NA       NA
##
## Maximum likelihood distribution is the limit of distributions
## with parameter vectors Estimate + s * GDOR as s goes to infinity
## (NA in the Estimate column should be read as zero).
```

The fact that we have `NA` for every parameter estimate means all parameters have been dropped to obtain identifiability in the LCM. Thus the LCM is completely degenerate (or we have complete separation in Agresti's terminology). The GDOR given is the direction the parameters go to infinity to get to the LCM. Since no parameters have been estimated, everything else is also `NA`.

Admittedly this is not easy to understand. But of course canonical parameters are never easy to understand.

But mean value parameters are much easier to understand.

```
estimates(lout, type = "mu")
```

```
## 1 2 3 4 5 6 7 8
## 0 0 0 0 1 1 1 1
```

This says the MLE of the saturated model mean value parameter vector μ is equal to the observed data. And since all of those values are on the boundary of the range (y_i can only be zero or one for these data), this means we have a completely degenerate LCM (something we already knew from the Estimates column in the summary being all `NA`).

10.4 So Now What?

Agresti (2013), Section 6.5.3 gives several pieces of advice about what to do in this situation. We now think all of it is bad, although we ourselves have given similar advice in the past. More on this below. None of this section is intended to be critical of Agresti (2013). That book is an authoritative compendium of conventional wisdom about the subject. The fact that conventional wisdom sometimes disagrees with long established theory, albeit fairly esoteric theory, is not unique to categorical data analysis.

The first thing we can do is wonder if some simpler model fits the data. If so, then we might not have to worry about this OM not having MLE. So let's try that.

```
lout.null <- llmdr(y ~ 1, family = "binomial")
anova(lout.null, lout)
```

```
## Analysis of Deviance Table
##
## Model 1: y ~ 1
## Model 2: y ~ x
##   Df as Null Df as Alt Deviance Df for test   LRT    P-value
## 1          1         0    11.09
## 2          0         2     0.00           1 11.09 0.00086778
```

We are getting a bit ahead of ourselves here (we will explain how hypothesis tests work in this situation in Section 14 below). But the P-value seems fairly strong. The response does depend on x. So we cannot drop x from the model.

Save these data for future use (Section about AIC below)

```
save.complete <- list(null = lout.null, alternative = lout,
    anova = anova(lout.null, lout), data = data.frame(x = x, y = y))
```

By the way, we could have gotten the same result from R function `glm` and friends.

```
gout.null <- glm(y ~ 1, family = "binomial")
anova(gout.null, gout, test = "LRT")
```

```
## Analysis of Deviance Table
##
## Model 1: y ~ 1
## Model 2: y ~ x
##   Resid. Df Resid. Dev Df Deviance  Pr(>Chi)
## 1         7      11.09
## 2         6       0.00  1    11.09 0.0008678 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This illustrates a general principle. When the MLE for the null hypothesis exists in the OM (of the null hypothesis), then the usual theory about likelihood ratio tests and Rao tests is valid. It doesn't matter whether or not the MLE exists in the OM of the alternative hypothesis (Section 14 below).

So R function `glm` and friends are doing the Right Thing here. But it gives a warning in fitting the alternative hypothesis that we have to be sophisticated enough to ignore (for the purposes of this hypothesis test).

Another thing Agresti recommends is to do likelihood-based confidence intervals. These at least will be automatically one sided. But R function `confint.glm` in R package `MASS` messes up one-sided confidence intervals. So we have to do that the hard way.

```
logl.factory <- function(modmat, response) {
    stopifnot(is.numeric(modmat))
    stopifnot(is.finite(modmat))
```

```
    stopifnot(is.matrix(modmat))
    stopifnot(is.numeric(response))
    stopifnot(is.finite(response))
    stopifnot(round(response) == response) # check for integer-valued
    stopifnot(response >= 0)
    stopifnot(length(response) == nrow(modmat))
    logpq <- function(theta) ifelse(theta > 0, - log1p(exp(- theta)),
            theta - log1p(exp(theta)))
    function(beta) {
        stopifnot(is.numeric(beta))
        stopifnot(is.finite(beta))
        stopifnot(length(beta) == ncol(modmat))
        theta <- drop(modmat %*% beta)
        logp <- logpq(theta)
        logq <- logpq(- theta)
        return(sum(response * logp + (1 - response) * logq))
    }
}
# need as.matrix because lout$x is sparse matrix class
logl <- logl.factory(as.matrix(lout$x), lout$y)

logl.gradient.factory <- function(modmat, response) {
    stopifnot(is.numeric(modmat))
    stopifnot(is.finite(modmat))
    stopifnot(is.matrix(modmat))
    stopifnot(is.numeric(response))
    stopifnot(is.finite(response))
    stopifnot(round(response) == response) # check for integer-valued
    stopifnot(response >= 0)
    stopifnot(length(response) == nrow(modmat))
    pq <- function(theta) ifelse(theta > 0, 1 / (1 + exp(- theta)),
            exp(theta) / (1 + exp(theta)))
    function(beta) {
        stopifnot(is.numeric(beta))
        stopifnot(is.finite(beta))
        stopifnot(length(beta) == ncol(modmat))
        theta <- drop(modmat %*% beta)
        p <- pq(theta)
        q <- pq(- theta)
        return(rbind(drop(crossprod(modmat, response * q - (1 - response) * p))))
    }
}
logl.gradient <- logl.gradient.factory(as.matrix(lout$x), lout$y)

beta.try <- rnorm(2)
logl.gradient(beta.try)
```

```
##      (Intercept)         x
## [1,]   -3.999691 -99.99691
```

```
grad(logl, beta.try)
```

```
## [1]  -3.999691 -99.996910
```

```
# check that maximum of logl is zero
logl(100 * lout$gdor)
```

```
## [1] -2.777589e-11
```

```
logl(300 * lout$gdor)
```

```
## [1] -5.357274e-33
```

So now we are going to do something really tricky that does not generalize to problems with more than two parameters but does really show what is going on in this example. We are going to draw the boundary of the likelihood-based confidence region for β. Because we are going to compare this with some other regions, we write a function to do it.

```
do.curve <- function(hin, hin.jac, eps = 0.1, cutoff = -10) {

    # find one point on curve
    beta.start <- c(0, 0)
    gdor <- lout$gdor
    aout <- auglag(beta.start,
        fn = function(beta) sum(gdor * beta),
        gr = function(beta) gdor,
        hin = hin, hin.jac = hin.jac,
        control.outer = list(trace = FALSE))
    stopifnot(aout$convergence == 0)

    beta.save <- aout$par

    # Now step along the curve in counterclockwise direction
    # using R function auglag to put us back on the curve
    for (ii in 1:1000) {
        foo <- logl.gradient(aout$par)
        # rotate clockwise 90 degrees
        bar <- rev(foo) * c(1, -1)
        # normalize
        bar <- bar / sqrt(sum(bar^2))

        # slowly increase step size as we go along
        beta.try <- aout$par + eps * bar * (1 + ii / 10)
        aout <- auglag(beta.try,
            fn = function(beta) sum((beta - beta.try)^2) / 2,
            gr = function(beta) beta - beta.try,
            hin = hin, hin.jac = hin.jac,
            control.outer = list(trace = FALSE))
        stopifnot(aout$convergence == 0)
        beta.save <- rbind(beta.save, aout$par)
        if (aout$par[1] < cutoff) break
    }

    # go back to the the first point
    aout <- list(par = beta.save[1, ])
    # and go the other way
    for (ii in 1:1000) {
        foo <- logl.gradient(aout$par)
        # rotate counterclockwise 90 degrees
```

```
        bar <- rev(foo) * c(-1, 1)
        # normalize
        bar <- bar / sqrt(sum(bar^2))

        # slowly increase step size as we go along
        beta.try <- aout$par + eps * bar * (1 + ii / 10)
        aout <- auglag(beta.try,
            fn = function(beta) sum((beta - beta.try)^2) / 2,
            gr = function(beta) beta - beta.try,
            hin = hin, hin.jac = hin.jac,
            control.outer = list(trace = FALSE))
        stopifnot(aout$convergence == 0)
        beta.save <- rbind(aout$par, beta.save)
        if (aout$par[1] < cutoff) break
    }
    beta.save
}
```

So that is a pretty complicated function, but now that we have it, it is fairly simple to use. Its arguments are

- `hin` the function whose zero set is the curve,
- `hin.jac` the gradient of that function,
- `eps` how far apart we want points along the curve to be, and
- `cutoff` where to stop. The curve, of course, goes off to infinity in the direction of recession at both ends, so we have to stop somewhere. Having made this plot in previous lecture notes (not in the same way, we know that the curve goes off the left side of the plot, so we stop when β_1 equals the cutoff.

This code (especially `cutoff`) is very specific to this particular problem, but since nothing of the sort works for more than two parameters, we won't worry about that.

```
level <- 0.95
# divide by 2 because our function is log likelihood not twice log likelihood
crit <- qchisq(level, df = length(lout$coefficients)) / 2
crit
```

```
## [1] 2.995732
```

```
beta.likelihood <- do.curve(function(beta) crit + logl(beta), logl.gradient)
```

```
foo <- beta.likelihood[,1] > -8
plot(beta.likelihood[,1], beta.likelihood[,2],
    xlab = expression(beta[1]), ylab = expression(beta[2]),
    xlim = range(beta.likelihood[foo,1]),
    ylim = range(beta.likelihood[foo,2]), type = "l")
```

Now we can make likelihood-based confidence intervals for any parameter (any function of β) by maximizing and minimizing it over this region. And all of these intervals will have simultaneous coverage because they are all based on the same confidence region.

If we want non-simultaneous coverage. We do the same thing except we use the critical value for one degree of freedom.

```
# divide by 2 because our function is log likelihood not twice log likelihood
crit <- qchisq(level, df = 1) / 2
crit
```

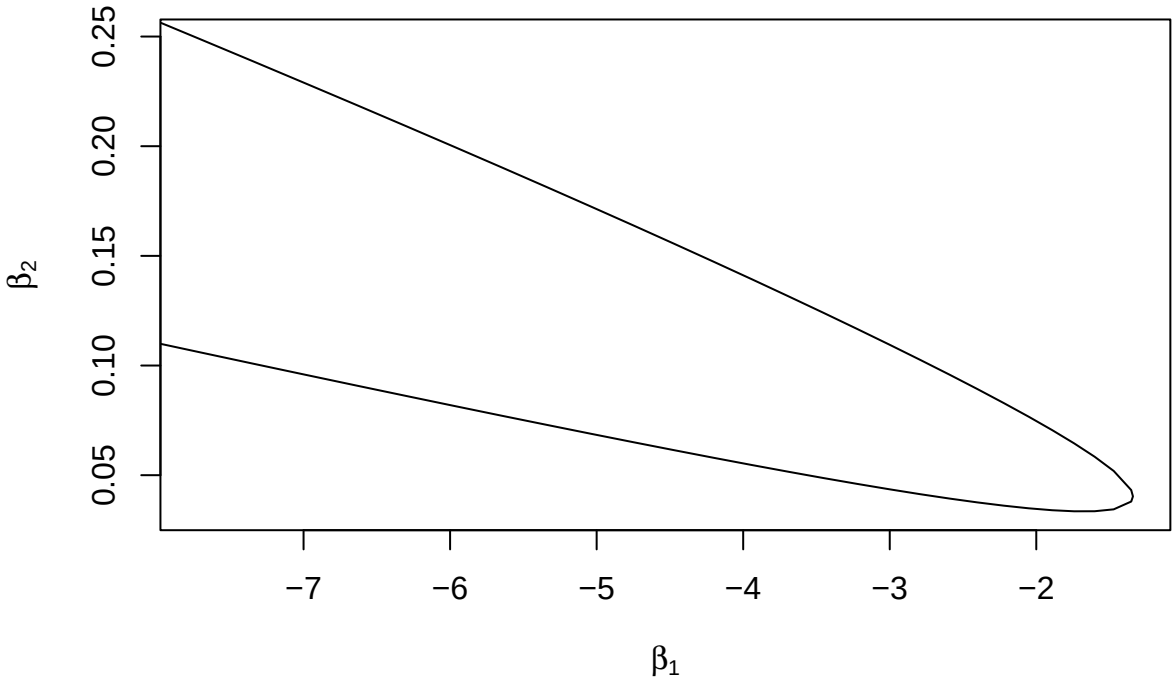


Figure 2: Likelihood-based Confidence Region for Beta

```
## [1] 1.920729
```

```
beta.likelihood.one.df <-
    do.curve(function(beta) crit + logl(beta), logl.gradient)
```

```
plot(beta.likelihood[,1], beta.likelihood[,2],
    xlab = expression(beta[1]), ylab = expression(beta[2]),
    xlim = range(beta.likelihood[foo,1]),
    ylim = range(beta.likelihood[foo,2]), type = "l")
lines(beta.likelihood.one.df[,1], beta.likelihood.one.df[,2], col = "red")
```

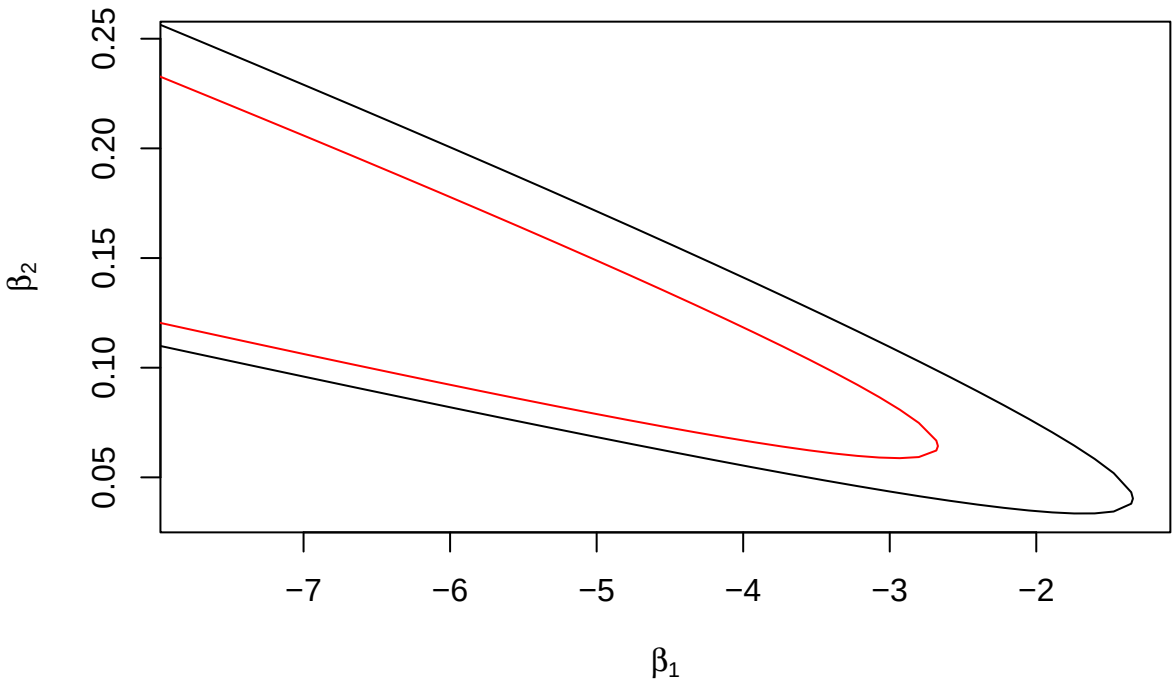


Figure 3: Level Sets of the Log Likelihood. Black, chi square critical value for 2 df. Red, chi square critical value for 1 df.

But there is a problem with what we are doing. It isn't justified by theory.

Let's go back to the binomial distribution. We know that for large n and π not close to the boundary of its parameter space that we have good normal approximation, which is what these likelihood based confidence intervals assume.

But we also know that when n is large and π is close to the boundary so either $n\pi$ or $n(1 - \pi)$ is not large, we have good Poisson approximation not normal approximation.

So that says that when the mean value parameter is close to the boundary, the usual asymptotics of maximum likelihood do not work. So there is every reason to believe these intervals just do not work.

Another way to explain the same thing is to note that the multivariate normal distribution has no boundary. Thus a sampling distribution is only approximately multivariate normal when the data for the distribution

being approximated are almost never on the boundary. Thus the mere fact that the MLE distribution is on the boundary is sufficient reason to believe the asymptotics do not work.

10.5 Valid Region for Beta

Thus we introduce another sort of confidence region, proposed by Geyer (2009), Section 3.16. These, at least, do have some theoretical justification. We won't explain that theory now, although it is quite simple (see Section 12 below). We will just put it too on our plot.

```
alpha <- 1 - level
alpha
```

```
## [1] 0.05
```

```
crit <- (- log(alpha))
crit
```

```
## [1] 2.995732
```

```
beta.geyer <- do.curve(function(beta) crit + logl(beta), logl.gradient,
    cutoff = -1000)
```

```
plot(beta.likelihood[,1], beta.likelihood[,2],
    xlab = expression(beta[1]), ylab = expression(beta[2]),
    xlim = range(beta.likelihood[foo,1]),
    ylim = range(beta.likelihood[foo,2]), type = "l")
lines(beta.likelihood.one.df[,1], beta.likelihood.one.df[,2], col = "red")
lines(beta.geyer[,1], beta.geyer[,2], col = "blue")
arrows(-5, 0.125, -5 + 2 * lout$gdor[1], 0.125 + 2 * lout$gdor[2])
```

The blue curve in the plot is exactly on top of the black curve. This is an accidental feature of this problem. It happens because the LCM is completely degenerate (so both the blue curve and the black curve use the same constraint function) and this problem has 2 parameters (which makes the critical values for the blue curve and the black curve the same). The blue and black curves would be different if the LCM were only partially degenerate or if the problem had number of parameters not equal to 2. If the number of parameters was much greater than 2, then the black curve would be far outside the blue curve, and even the red curve might be far outside the blue curve.

So we will henceforth discard the likelihood-based intervals as a bad idea (when near the boundary in the mean value parameter space and near infinity in the canonical parameter space; likelihood-intervals are one of the great ideas of statistics when far from the boundary) and use the proposal of Geyer (2009).

10.6 GDOR Need Not Be Unique

In this problem there are an infinite number of directions of recession. R package `rcdd` can find all of them.

```
tanv <- as.matrix(lout$x)
tanv[lout$y == 1, ] <- (- tanv[lout$y == 1, ])
vrep <- makeV(rays = tanv)
hrep <- scdd(vrep)$output
norv <- (- hrep[ , -(1:2), drop = FALSE])
norv
```

```
##      [,1] [,2]
## [1,]  -60    1
## [2,]  -40    1
```

The rows of this matrix are the extreme directions of recession. They are not generic. Every vector pointing in a direction between these vectors (every vector that is a linear combination of these vectors with strictly

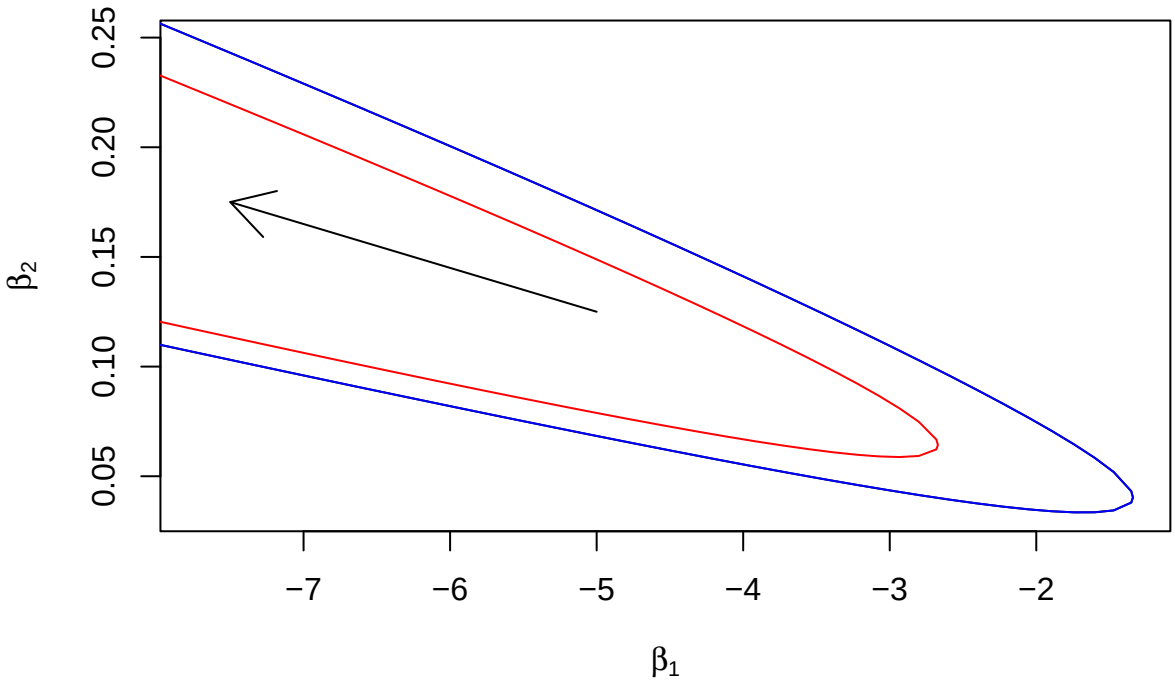


Figure 4: Level Sets of the Log Likelihood. Black, chi square critical value for 2 df. Red, chi square critical value for 1 df. Blue, minus log significance level. Arrow is GDOR. Blue curve is on top of black curve.

positive coefficients) is a GDOR. Let's put these on our plot.

```
plot(beta.likelihood[,1], beta.likelihood[,2],
    xlab = expression(beta[1]), ylab = expression(beta[2]),
    xlim = range(beta.likelihood[foo,1]),
    ylim = range(beta.likelihood[foo,2]), type = "n")
lines(beta.geyer[,1], beta.geyer[,2], col = "blue")
arrows(-5, 0.125, -5 + 2 * lout$gdor[1], 0.125 + 2 * lout$gdor[2])
norv <- norv / norv[1, 1] * lout$gdor[1]
arrows(-5, 0.125, -5 + 2 * norv[1, 1], 0.125 + 2 * norv[1, 2], col = "green3")
norv <- norv / norv[2, 1] * lout$gdor[1]
arrows(-5, 0.125, -5 + 2 * norv[2, 1], 0.125 + 2 * norv[2, 2], col = "green3")
```

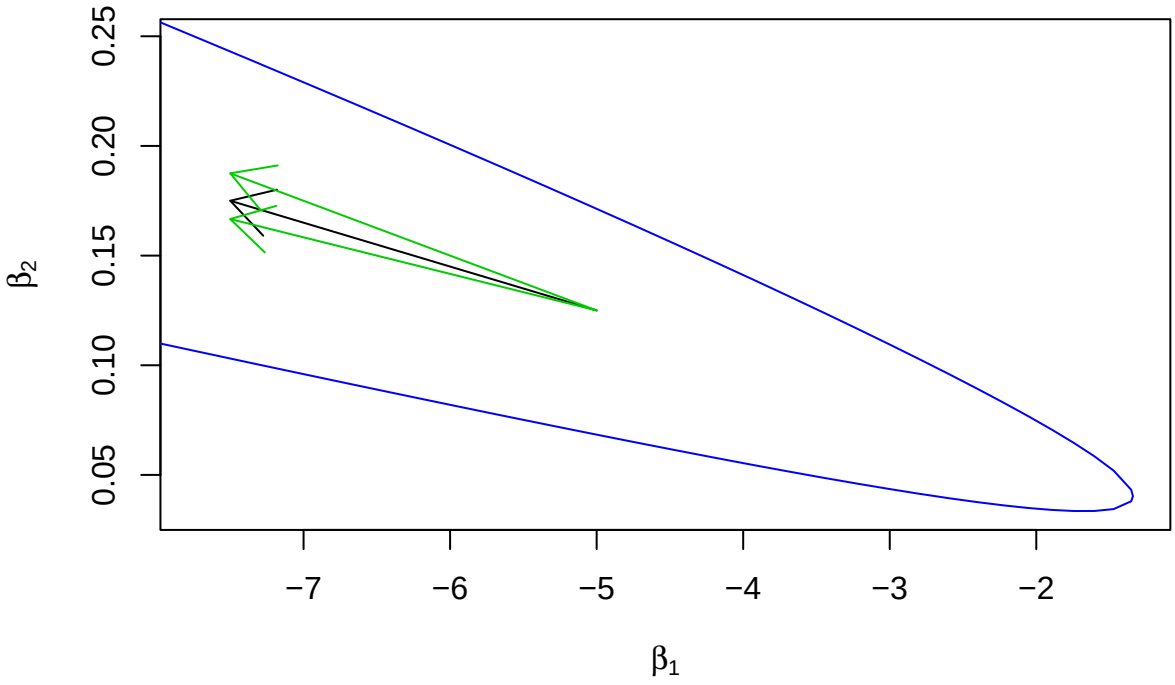


Figure 5: Level Set of Log Likelihood and Directions of Recession. Blue curve is boundary of level set (same as blue curve in Figure
reffig:agresti-complete-geyer-plot). Black arrow is GDOR. Green arrows are DOR that are not generic.

Any arrow pointing between the green arrows is a GDOR, so GDOR are not unique in this example (they are unique in a slightly different example below).

There are two reasons why we don't try to find all GDOR rather than just one GDOR.

- One GDOR is enough to do all the statistics.
- All the GDOR take a lot longer to find, hours, days, or worse for large examples.

10.7 Mathematical Proofs

The arrow in Figure 4 or the arrows in Figure 5 show that the log likelihood really does go uphill all the way to infinity in the GDOR direction. Of course, just eyeballing a plot isn't a proof, but the computer can do a rigorous proof.

```
lout <- llmdr(y ~ x, family = "binomial", proofs = 2)
lout$proofs
```

```
## [1] 2
```

The argument `proofs = 2` to R function `llmdr` asks it to do 2 proofs (more precisely 2 parts of one proof). The result having `lout$proofs == 2` indicates that both proofs were successful. The proofs have been done. What is reported as a GDOR really is one. (Or for a problem in which a GDOR does not exist so the MLE exists in the OM, that has been proved.)

By reporting it has done the proofs, the computer says that, although it found the answer using inexact computer arithmetic, it went back and did enough work in infinite-precision rational arithmetic to prove that what it says is a GDOR really is one.

The only reason R function `llmdr` does not do these proofs by default is that inexact computer arithmetic usually gives the right answer and these proofs can take many minutes to do on large problems.

10.8 Intervals for Beta

```
beta <- beta.geyer
colnames(beta) <- names(lout$coefficients)
beta <- apply(beta, 2, range)
rownames(beta) <- c("lower", "upper")
beta <- t(beta)
beta
```

```
##                      lower    upper
## (Intercept) -1.001834e+03 -1.34039
## x            3.354413e-02 25.11505
```

Except we know this is wrong. As the arrow in Figure 4 shows, β_1 goes to $-\infty$ as we travel in the GDOR direction, and β_2 goes to $+\infty$ as we travel in the GDOR direction. We just don't get that far on the plot. So fix that up.

```
beta[1, "lower"] <- -Inf
beta[2, "upper"] <- Inf
beta
```

```
##                  lower    upper
## (Intercept)       -Inf -1.34039
## x           0.03354413      Inf
```

We can make these intervals if a user asks for them, but they are not very informative. They provide much less information than the whole region whose boundary is the blue curve in Figure 4.

10.9 Intervals for Theta and Mu

Larry Brown (the author of Brown (1986)) told your humble author that nobody would ever understand the intervals for β so focus on μ. Your humble author mostly agrees. The only reasons we bothered with β is because

- everything starts there, we can only do intervals for other parameters by mapping intervals for β to them,

- when writing an R package, you don't know what users will want or what applications they will be doing, so you have to provide all the possibilities you can if you want to be useful.

First we do sloppy intervals for θ.

```
modmat <- as.matrix(lout$x)
theta.geyer <- modmat %*% t(beta.geyer)
theta.intervals <- apply(theta.geyer, 1, range)
theta.intervals <- t(theta.intervals)
colnames(theta.intervals) <- c("lower", "upper")
theta.intervals
```

```
##          lower        upper
## 1 -835.3521315   -0.9200953
## 2 -668.8705930   -0.4331595
## 3 -502.3890545    0.2852195
## 4 -335.9075160    2.9444390
## 5   -2.9444390  505.2454046
## 6   -0.2850279  756.3958874
## 7    0.4304010 1007.5463703
## 8    0.9193835 1258.6968531
```

But we can do a lot better than this by running the optimizer again. First we know that we have one-sided intervals. If $y_i = 0$, then we know the lower bound for θ_i is $-\infty$. If $y_i = 1$, then we know the upper bound for θ_i is $+\infty$. So we can just set them without any calculation.

Then we find the β that optimizes each θ_i and use it as a starting point for optimization. And we write a function to do that.

```
do.theta.intervals <- function(x) {
    foo <- cbind(1, x) %*% t(beta.geyer)
    gdor <- lout$gdor

    lower <- rep(-Inf, length(x))
    upper <- rep(+Inf, length(x))
    for (i in seq(along = x)) {
        eta <- sum(c(1, x[i]) * gdor)
        eta <- zapsmall(eta)
        if (eta < 0) {
            # need upper bound
            idx <- which(foo[i, ] == max(foo[i, ]))
            idx <- idx[1] # just in case of ties
            beta.start <- beta.geyer[idx, ]
            aout <- auglag(beta.start,
                fn = function(beta) sum(c(1, x[i]) * beta),
                gr = function(beta) c(1, x[i]),
                hin = function(beta) crit + logl(beta),
                hin.jac = logl.gradient,
                control.outer = list(trace = FALSE),
                control.optim = list(fnscale = -1))
            stopifnot(aout$convergence == 0)
            upper[i] <- aout$value
        } else {
            # need lower bound
            idx <- which(foo[i, ] == min(foo[i, ]))
            idx <- idx[1] # just in case of ties
            beta.start <- beta.geyer[idx, ]
```

```
            aout <- auglag(beta.start,
                fn = function(beta) sum(c(1, x[i]) * beta),
                gr = function(beta) c(1, x[i]),
                hin = function(beta) crit + logl(beta),
                hin.jac = logl.gradient,
                control.outer = list(trace = FALSE))
            stopifnot(aout$convergence == 0)
            lower[i] <- aout$value
        }
    }
    cbind(lower, upper)
}
theta <- do.theta.intervals(x)
theta
```

```
##           lower      upper
## [1,]       -Inf -0.9198396
## [2,]       -Inf -0.4331413
## [3,]       -Inf  0.2852195
## [4,]       -Inf  2.9444390
## [5,] -2.9444390        Inf
## [6,] -0.2850279        Inf
## [7,]  0.4304010        Inf
## [8,]  0.9193829        Inf
```

And then we transform these to the mean value parameter scale.

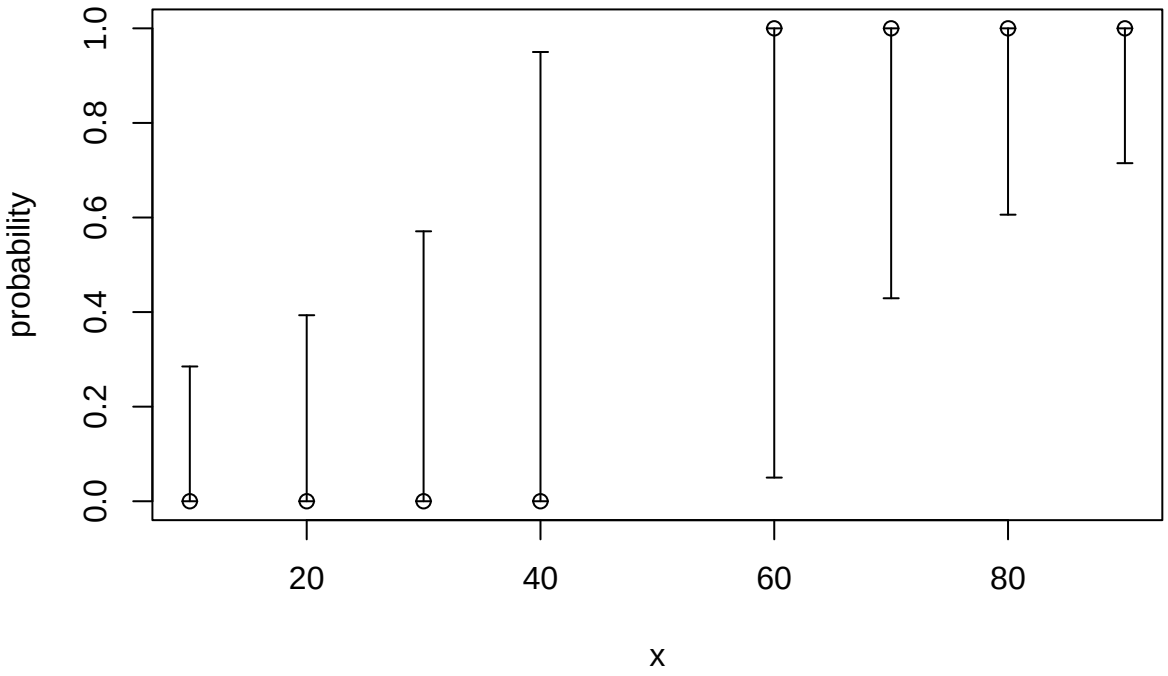
```
mu <- 1 / (1 + exp(- theta))
mu
```

```
##          lower     upper
## [1,] 0.0000000 0.2849906
## [2,] 0.0000000 0.3933765
## [3,] 0.0000000 0.5708254
## [4,] 0.0000000 0.9500000
## [5,] 0.0500000 1.0000000
## [6,] 0.4292216 1.0000000
## [7,] 0.6059694 1.0000000
## [8,] 0.7149164 1.0000000
```

And then we plot them.

```
errbar(x, y, mu[ , 2], mu[ , 1], xlab = "x", ylab = "probability")
```

\begin{figure}



{

}

\caption{Simultaneous 95% Confidence intervals for Mean Value Parameters. Hollow circles are MLE.}
\end{figure}

This plot is fairly simple. The hollow dots are the MLE saturated model mean value parameter vector $\hat{\mu}$. They are on the boundary. The MLE distribution is completely degenerate and says the observed value of the response vector is the only possible value.

But $\hat{\mu}$ is not μ. Estimates are not the parameters they estimate. This is the second most fundamental concept in statistics. (The most fundamental is anecdotes are not data. The third most fundamental is it is called a 95% confidence interval because it is wrong 5% of the time.)

So we see that we do not need to fear MLE on the boundary. We just need a proper theory of confidence intervals that applies in the situation.

Note that the error bars say almost nothing about the probabilities near the jump in the regression function. Not surprising in hindsight. If one of the Bernoulli random variables for $x = 40$ or $x = 60$ had been different, we would have inferred the jump was at a different point. But the farther we get from the jump the better the estimation.

So even in this toy problem with so little data, statistics can say something of consequence.

10.10 Submodel Canonical Statistic

Theory tells us that where the action is really happening is with the canonical statistic vector $M^T y$ of the canonical affine submodel.

In most problems we can't visualize that, but in this toy problem, where $M^T y$ is two-dimensional, we can.

Another reason why we can't do anything like this section in non-toy problems is that the computing we do in this section is exponential in the size of the data, so the computing time would be horrendously long for non-toy data.

There are 2^n possible values of the response vector where n is the length of the response vector because each component of y can be either zero or one. The following code makes all of those vectors.

```
yy <- matrix(0:1, nrow = 2, ncol = length(x))
colnames(yy) <- paste0("y", x)
yy <- expand.grid(as.data.frame(yy))

head(yy)
```

```
##   y10 y20 y30 y40 y60 y70 y80 y90
## 1   0   0   0   0   0   0   0   0
## 2   1   0   0   0   0   0   0   0
## 3   0   1   0   0   0   0   0   0
## 4   1   1   0   0   0   0   0   0
## 5   0   0   1   0   0   0   0   0
## 6   1   0   1   0   0   0   0   0
```

```
dim(yy)
```

```
## [1] 256   8
```

But there are not so many distinct values of the submodel canonical statistic.

```
m <- cbind(1, x)
mtyy <- t(m) %*% t(yy)
t1 <- mtyy[1, ]
t2 <- mtyy[2, ]
t1.obs <- sum(y)
t2.obs <- sum(x * y)
```

```
t.key <- paste(t1, t2, sep = ":")
t.idx <- match(unique(t.key), t.key)
t1.uniq <- t1[t.idx]
t2.uniq <- t2[t.idx]

par(mar = c(5, 5, 1, 0) + 0.1)
plot(t1.uniq, t2.uniq, xlab = expression(t[1] == sum(y)),
    ylab = expression(t[2] == sum(x * y)))
points(t1.obs, t2.obs, pch = 19)

# add H
gdor <- lout$gdor
# rotate 90 degrees to convert normal vector to tangent vector
tanv <- rev(gdor) * c(-1, 1)
tanv
```

```
##           x (Intercept)
##      -0.025      -1.250
```

```
# now convert to slope and intercept of line
slope <- tanv[2] / tanv[1]
intercept <- t2.obs - t1.obs * slope
abline(intercept, slope)
```

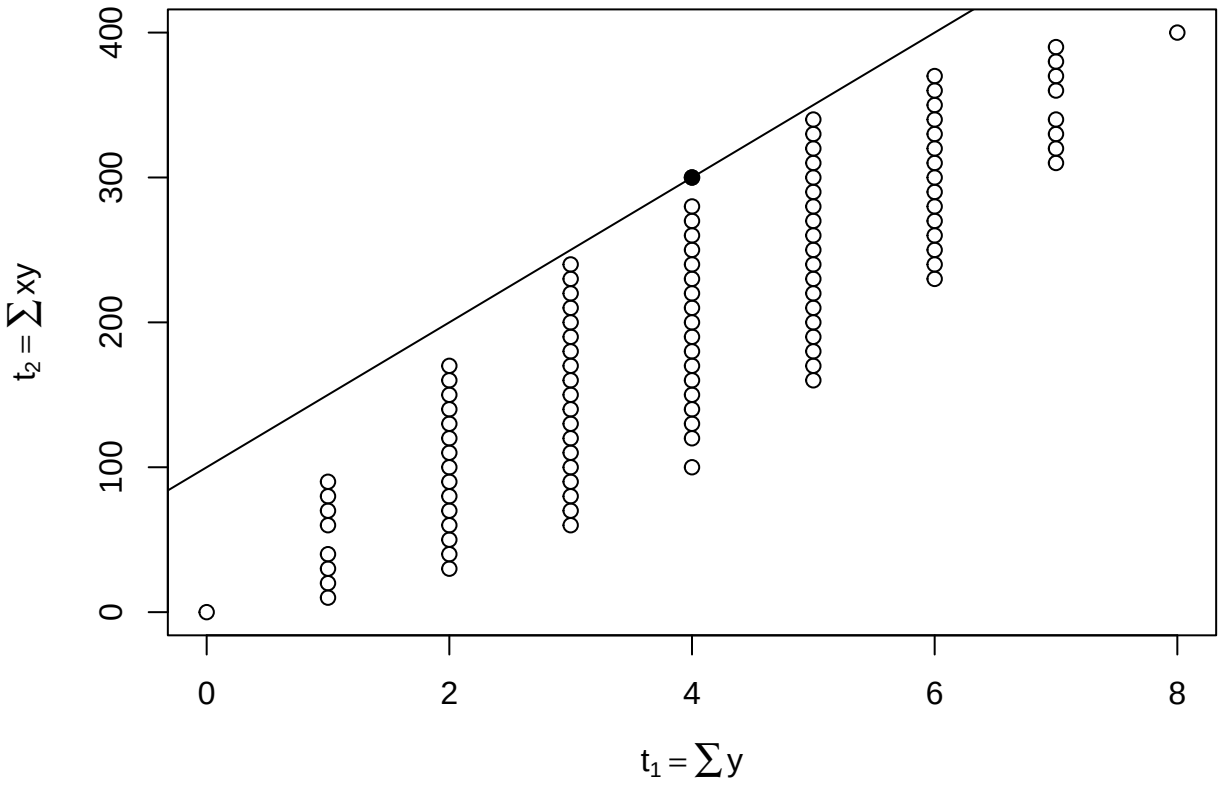


Figure 6: Possible values of the submodel canonical statistic vector $M^T y$ for the data shown in Figure reffig:one. Solid dot is the observed value. Line is hyperplane eqrefeq:h determined by the GDOR.

Having the hyperplane H defined by (1) on the plot makes it clear that the observed value of the canonical statistic really is extreme. It lies on the line, and all other possible (in the OM) values lie on one side of H. The fact that only one point lies in H means the LCM is concentrated at this one point, so is completely degenerate.

10.11 Complete Separation

Just for completeness, we do show the "complete separation" plot that Agresti discusses.

```
dim(modmat)
```

```
## [1] 8 2
```

```
plot(modmat[ , 1], modmat[ , 2],
    xlab = colnames(modmat)[1], ylab = colnames(modmat)[2])
points(modmat[y == 1, 1], modmat[y == 1, 2], pch = 19)
intercept <- 50 - 1 * slope
abline(intercept, slope)
```

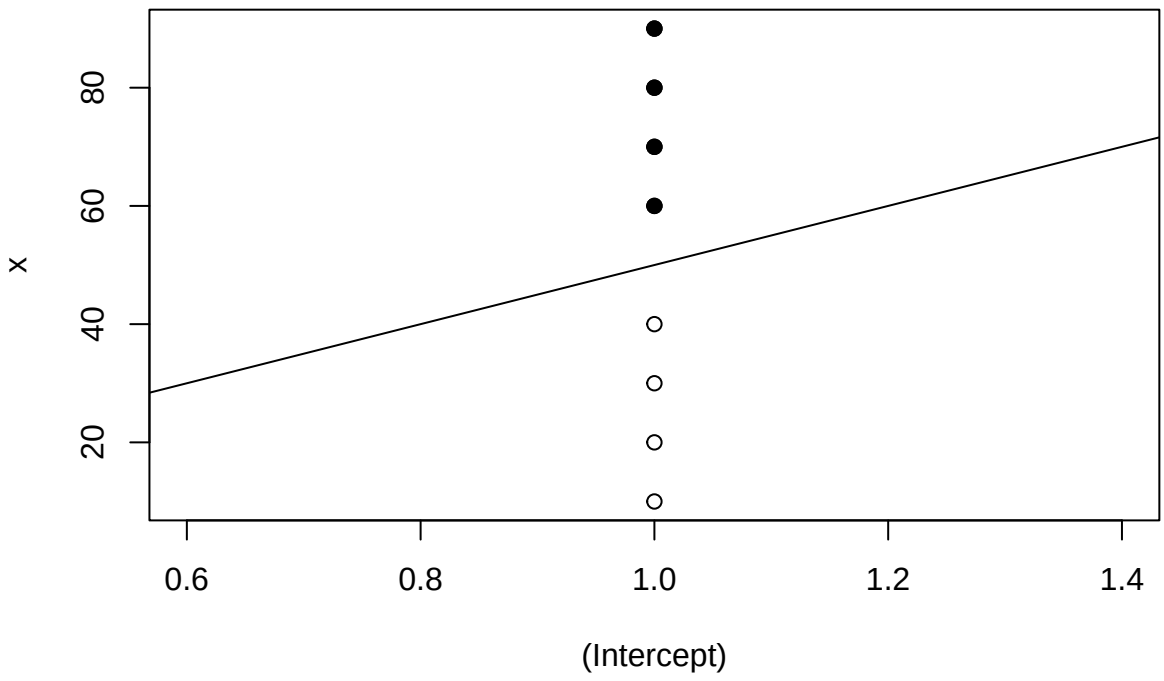


Figure 7: Complete Separation of Successes and Failures. Black dots successes. White dots failures. Black line separating hyperplane

The separating hyperplane in Figure 7 is one whose normal vector is the GDOR found by R function `llmdr`. It is clear that there are a lot of separating hyperplanes, so this plot, although easier to do than Figure 6, doesn't provide the same information. This will become even more clear in the following section.

Figure 7 is a little odd in that the point cloud isn't really two-dimensional. All of the points have the same value of the regressor `(Intercept)`. But of course that always happens. That it what "intercept" means.

So it is odd, but also extremely common. Most models have an intercept.

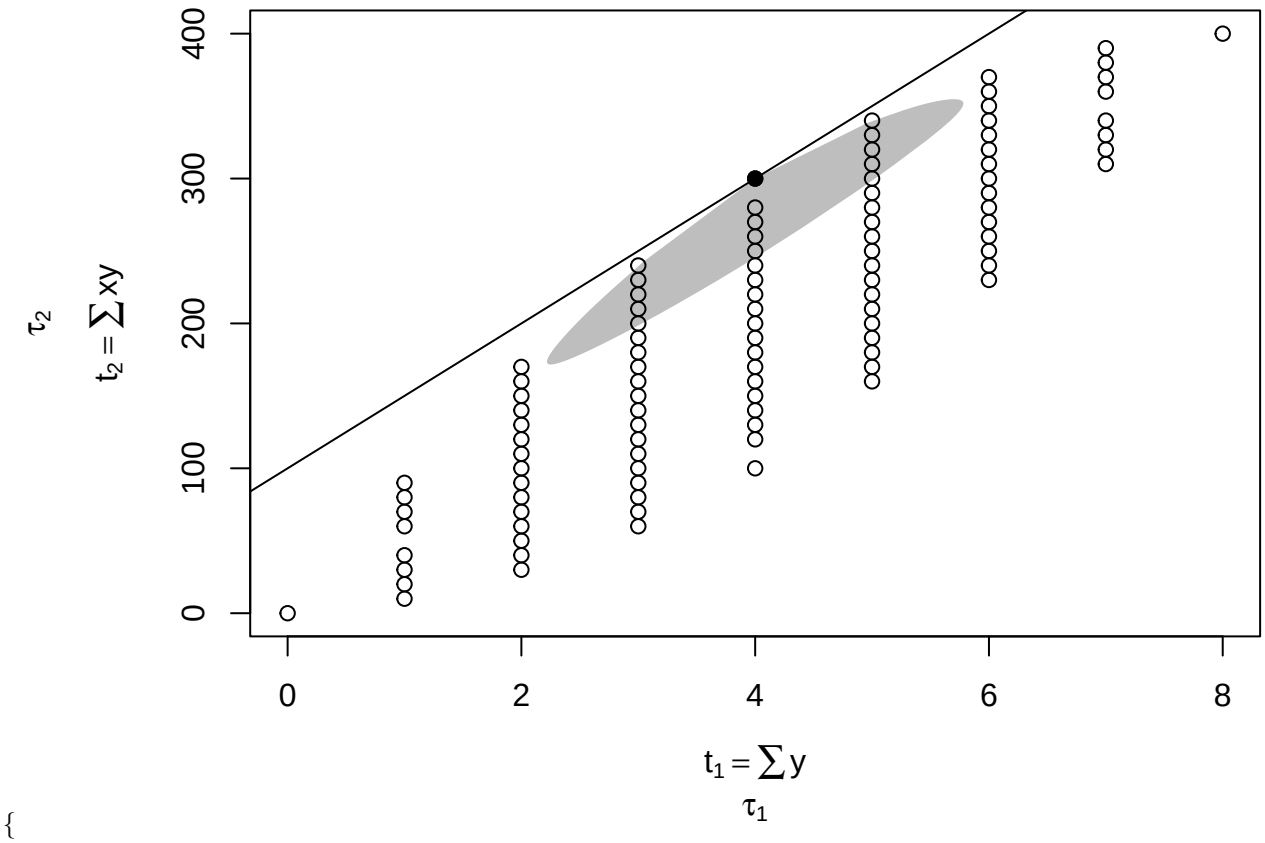
10.12 Region for Tau

We can map the confidence region for β that is the region bounded by the blue curve in Figure 4 to values for τ thus obtaining a confidence region for τ.

```
theta <- tcrossprod(modmat, beta.geyer)
mu <- 1 / (1 + exp(- theta))
tau <- crossprod(modmat, mu)
# add observed value to these points
tau <- cbind(c(t1.obs, t2.obs), tau)

par(mar = c(5, 6, 1, 0) + 0.1)
plot(t1.uniq, t2.uniq, xlab = expression(t[1] == sum(y)),
    ylab = expression(t[2] == sum(x * y)), type = "n")
polygon(tau[1, ], tau[2, ], border = NA, col = "gray")
points(t1.uniq, t2.uniq)
points(t1.obs, t2.obs, pch = 19)
intercept <- t2.obs - t1.obs * slope
abline(intercept, slope)
mtext(expression(tau[1]), side = 1, line = 4)
mtext(expression(tau[2]), side = 2, line = 5)
```

\begin{figure}



{

}

\caption{Same as Figure 6 Except Gray Region is 95% Confidence Region for Submodel Mean Value Parameter} \end{figure}

From this figure it is clear that the confidence region is fairly restrictive, and most of the submodel mean value parameter space is excluded by the confidence region.

10.13 Confidence Interval for Closeness to the Boundary

We can make confidence intervals for any function of parameters we can think of. Here is one that is especially interesting.

The parameter $\langle \tau, \delta \rangle$ measures how close a distribution is to the boundary (to the line in the figure).

```
sum(c(t1.obs, t2.obs) * gdor) - rev(range(crossprod(tau, gdor)))
```

```
## [1] 0.000000 1.363626
```

Admittedly, this is impossible to interpret by itself because the length of the GDOR is arbitrary. But we can compare it to the range of the same GDOR applied to the whole sample space.

```
ttt <- rbind(t1.uniq, t2.uniq)
sum(c(t1.obs, t2.obs) * gdor) - rev(range(crossprod(ttt, gdor)))
```

```
## [1] 0 5
```

These data, few though they may be, do mush up the probability fairly close to the boundary.

10.14 Confidence Intervals for Hypothetical Individuals

We can also make confidence intervals for hypothetical individuals, what R generic function `predict` does when its `newdata` argument is used.

Here we treat x as a continuous variable and do intervals for μ for all possible x values.

```
xx <- seq(0, 100, 0.1)
# be especially careful at jump
xx <- c(xx, 39.99, 40.01, 59.99, 60.01)
xx <- sort(xx)
theta <- do.theta.intervals(xx)
mu <- 1 / (1 + exp(- theta))
# have to do by hand since optimization isn't good enough
mu[xx < 60, 1] <- 0
mu[xx > 40, 2] <- 1
```

```
plot(x, y, type = "n",
    xlab = expression(x), ylab = expression(mu(x)))
xxx <- c(xx, rev(xx))
yyy <- c(mu[ , 1], rev(mu[ , 2]))
length(xxx); length(yyy)
```

```
## [1] 2010
```

```
## [1] 2010
```

```
polygon(xxx, yyy, border = NA, col = "gray")
points(x, y)
```

That finishes our work on this toy problem. Admittedly, it is hard to know what you are supposed to learn from a toy problem. But toy problems do have the advantage of at least being somewhat understandable.

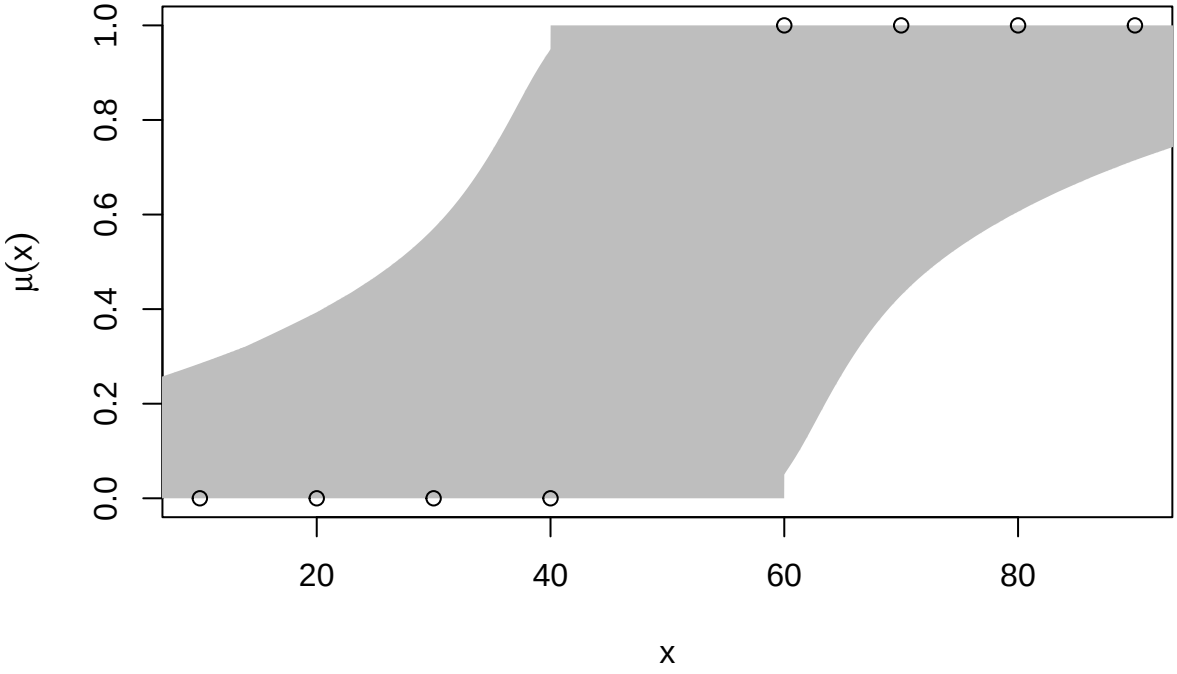


Figure 8: Like Figurereffig:agresti-complete-mu-plot Except with Confidence Intervals for Every x value

11 A Clinical Trial Example of Agresti

But analyses involving DOR are not always so complicated. Sometimes the issue and analysis are really quite simple.

Section 6.5.2 in Agresti (2013) looks at the following data for a clinical trial.

```
center <- rep(1:5, each = 2)
center <- as.factor(center)
treatment <- rep(c("active_drug", "placebo"), times = 5)
success <- c(0, 0, 1, 0, 0, 0, 6, 2, 5, 2)
failure <- c(5, 9, 12, 10, 7, 5, 3, 6, 9, 12)
y <- cbind(success, failure)
```

If we dump that into R function `llmdr` we find the MLE does not exist in the OM.

```
lout <- llmdr(y ~ center + treatment, family = "binomial")
summary(lout)
```

```
##
## Call:
## llmdr(formula = y ~ center + treatment, family = "binomial")
##
##                   Estimate    GDOR Std. Error z value Pr(>|z|)
## (Intercept)       -0.4763 -1.0000     0.5058  -0.942   0.3463
## center2           -2.1802  1.0000     1.1327  -1.925   0.0543 .
## center3                NA  0.0000         NA      NA       NA
## center4            1.0631  1.0000     0.7011   1.516   0.1294
## center5                NA  1.0000         NA      NA       NA
## treatmentplacebo  -1.5460  0.0000     0.7017  -2.203   0.0276 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Maximum likelihood distribution is the limit of distributions
## with parameter vectors Estimate + s * GDOR as s goes to infinity
## (NA in the Estimate column should be read as zero).
```

Furthermore, we cannot drop `treatment` because it is statistically significant (0.0276) by the Wald test done in the summary, and we cannot drop `center` because

```
lout.null <- llmdr(y ~ treatment, family = "binomial")
anova(lout.null, lout)
```

```
## Analysis of Deviance Table
##
## Model 1: y ~ treatment
## Model 2: y ~ center + treatment
##   Df as Null Df as Alt Deviance Df for test    LRT    P-value
## 1          2         0  23.9266
## 2          4         6   0.5021           4 23.424 0.00010415
```

So the model `y ~ center + treatment` is the only model under consideration that fits the data.

As usual it is very hard to figure out what canonical parameters mean. So we look at mean value parameters.

```
data.frame(center, treatment, estimates(lout, "mu"))
```

```
##    center   treatment   success   failure
```

```
## 1       1 active_drug 0.0000000  5.000000
## 2       1     placebo 0.0000000  9.000000
## 3       2 active_drug 0.8526289 12.147371
## 4       2     placebo 0.1473711  9.852629
## 5       3 active_drug 0.0000000  7.000000
## 6       3     placebo 0.0000000  5.000000
## 7       4 active_drug 5.7836818  3.216318
## 8       4     placebo 2.2163182  5.783682
## 9       5 active_drug 5.3636893  8.636311
## 10      5     placebo 1.6363107 12.363689
```

Oh! There were no successes in either treatment group in centers 1 and 3. And the MLE mean values also say no successes in those centers.

If you were to decide to throw away the data from centers 1 and 3, just to make the data easier for you to analyze, then you could be severely criticized for that. If you did it and didn't admit it, that would be fraud. If you did it and did admit it, then that would be sufficient reason for everybody to ignore everything you say on the subject.

But if statistics does the same thing, following the theory of maximum likelihood estimation in exponential families, then there is nothing to criticize. The MLE in the LCM is also the MLE in the completion of the OM, arrived at by taking limits of sequences of distributions in the OM. There is nothing wrong with that. If taking limits is wrong, then everything involving calculus has been wrong for over 400 years. So that argument is a non-starter.

So having arrived at this conclusion, we mention that this is another situation in which the theory in these notes matches conventional practice. If a conclusion can be drawn solely from the fit in the LCM alone (with no reference to the OM), then it is valid (assuming the usual asymptotics of maximum likelihood apply, that is, assuming the sample size of the LCM is large enough).

So that means the coefficient for the treatment effect and its standard error in the summary above are valid. So we can turn that into a valid Wald confidence interval.

```
sout <- summary(lout)
fit <- sout$coefficients["treatmentplacebo", "Estimate"]
se.fit <- sout$coefficients["treatmentplacebo", "Std. Error"]
fit + c(-1, 1) * qnorm((1 + level) / 2) * se.fit
```

```
## [1] -2.9212133 -0.1707793
```

And the hypothesis test is valid too. We have a statistically significant treatment effect (with aforementioned P-value $P = 0.0276$). And this agrees with the corresponding confidence interval not containing zero (as it has to by the duality of hypothesis tests and confidence intervals).

Of course this does ignore the data from centers 1 and 3. But that is what maximum likelihood does with these data. Once we decide that `center` needs to be in the model, then we have to let statistics do what it does.

Of course, if we were interested in confidence intervals for centers 1 and 3, then we would have to do calculations like those in the preceding section. But as long as we are happy with what has been done in this section, then we're good.

By the way

```
gout <- glm(y ~ center + treatment, family = "binomial")
summary(gout)
```

```
##
## Call:
## glm(formula = y ~ center + treatment, family = "binomial")
```

```
##
## Coefficients:
##                     Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -2.305e+01  2.330e+04  -0.001   0.9992
## center2           2.039e+01  2.330e+04   0.001   0.9993
## center3           4.809e-03  3.172e+04   0.000   1.0000
## center4           2.363e+01  2.330e+04   0.001   0.9992
## center5           2.257e+01  2.330e+04   0.001   0.9992
## treatmentplacebo -1.546e+00  7.017e-01  -2.203   0.0276 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 28.53202  on 9  degrees of freedom
## Residual deviance:  0.50214  on 4  degrees of freedom
## AIC: 24.859
##
## Number of Fisher Scoring iterations: 21
```

gives a false negative by failing to warn `fitted probabilities numerically 0 or 1 occurred`.

It does agree with our P-value for the treatment effect. But that is an accident of having orthogonal predictors. If `treatment` and `center` were correlated predictors, then we wouldn't get the same answers.

12 Theory of Confidence Intervals

12.1 Confidence Intervals for OM when there is a GDOR

These are the intervals based on the region bounded by the blue curve in Figure 4. We have already met them. Now for the details.

Suppose the GDOR δ were known in advance of obtaining the data (this is not true; the GDOR depends on the data), and suppose we chose $\langle Y, \delta \rangle$ as a test statistic for a hypothesis test with a point null hypothesis β. We observe the largest possible value of this test statistic $\langle y, \delta \rangle$. Thus the P-value of this test is

$$\text{pr}_\beta(\langle Y, \delta \rangle = \langle y, \delta \rangle) = \text{pr}_\beta(Y \in H)$$

where H is given by (1). If we think decision theoretically, we reject the null hypothesis when this P-value is less than the significance level, call that α.

When we invert a hypothesis test to get a confidence region, the confidence region is the set of parameter values considered as null hypotheses that are not rejected by the hypothesis test. Of course, the probability that the null hypothesis is not rejected is $1 - \alpha$, so that is the coverage probability of the confidence region. In short, our confidence region is the set of β such that

$$\text{pr}_\beta(Y \in H) \geq \alpha \tag{2}$$

It makes no sense to use confidence regions of this kind when the MLE exists in the OM. Then this confidence region does nothing at all because saying there is no GDOR is equivalent to saying $\delta = 0$, hence H is the whole vector space where the canonical statistic lives, hence the region satisfying (2) is the whole parameter space.

12.2 Confidence Intervals for LCM when it is Not Completely Degenerate

When the LCM is not completely degenerate, it is an exponential family for which the MLE exists and the usual asymptotics may hold (if there is enough data after conditioning to make the LCM). So we use that.

Let l_{LCM} be the log likelihood function for the LCM. Then we make a confidence region that is the set of β such that

$$2[l_{\text{LCM}}(\hat{\beta}) - l_{\text{LCM}}(\beta)] \le \text{crit} \tag{3}$$

where crit is chosen to be an appropriate chi-square critical value with degrees of freedom chosen to give the desired coverage, and where in case the MLE exists in the OM we say LCM = OM, so the log likelihood in (3) is the log likelihood of the OM. One uses degrees of freedom equal to the number of identifiable parameters in the LCM to get simultaneous coverage. One uses degrees of freedom equal to one if one doesn't care about simultaneous coverage.

So much, so conventional. But things get a little stranger when we use the same parameterization for both of these kinds of confidence regions. If β is the parameter of the OM, for both of these, then the region satisfying (3) is also unbounded. The GDOR of the OM, at least, is a DOC of the LCM, so the region satisfying (3) is also unbounded in the direction of the GDOR and also in the opposite direction, and perhaps other directions (the LCM can have more DOC than the GDOR of the LCM).

It makes no sense to use confidence regions of this kind when the LCM is completely degenerate. Then this confidence region does nothing at all because the LCM contains only one distribution, so every direction is a DOC of the LCM and l_{LCM} is a constant function of β, hence the region satisfying (3) is the whole parameter space.

12.3 The Combination of the Two

To use both kinds of information, we use the confidence region consisting of β that satisfy both (2) and (3) except when the MLE exists in the OM, when we take the region satisfying (3) only, or except when the LCM is completely degenerate, when we take the region satisfying (2) only.

In case the LCM is partially degenerate and we are using both kinds of confidence regions, we have to correct for multiple testing. Because (2) only involves the probability of the event $Y \in H$ and (3) only involves the probability distribution of Y given the event $Y \in H$, probabilities multiply (joint equals marginal times conditional). Thus if the confidence level for the region satisfying (3) has confidence level $1 - \alpha_1$ and the confidence level for the region satisfying (2) has confidence level $1 - \alpha_2$, then the confidence level of the combination is $(1 - \alpha_1)(1 - \alpha_2)$. And we need to adjust α_1 and α_2 to get the desired level. Usually we just take both $1 - \alpha_1$ and $1 - \alpha_2$ to be the square root of the desired level, except in case the MLE exists in the OM and we take $1 - \alpha_1$ to be the desired level or in case the LCM is completely degenerate and we take $1 - \alpha_2$ to be the desired level.

12.4 Confidence Regions to Confidence Intervals

For any function $g(\beta)$ we can form a confidence interval for this parameter by minimizing and maximizing $g(\beta)$ over the region satisfying both (2) and (3).

13 Quasi-Complete Separation Example of Agresti

13.1 Data

Section 6.5.1 of Agresti (2013) introduces the notion of quasi-complete separation with the following example.

```
x <- seq(10, 90, 10)
x <- c(x, 50)
y <- as.numeric(x > 50)
y[x == 50] <- c(0, 1)
data.frame(x, y)
```

```
##     x y
## 1  10 0
## 2  20 0
## 3  30 0
## 4  40 0
## 5  50 0
## 6  60 1
## 7  70 1
## 8  80 1
## 9  90 1
## 10 50 1
```

Figure 9 shows these data.

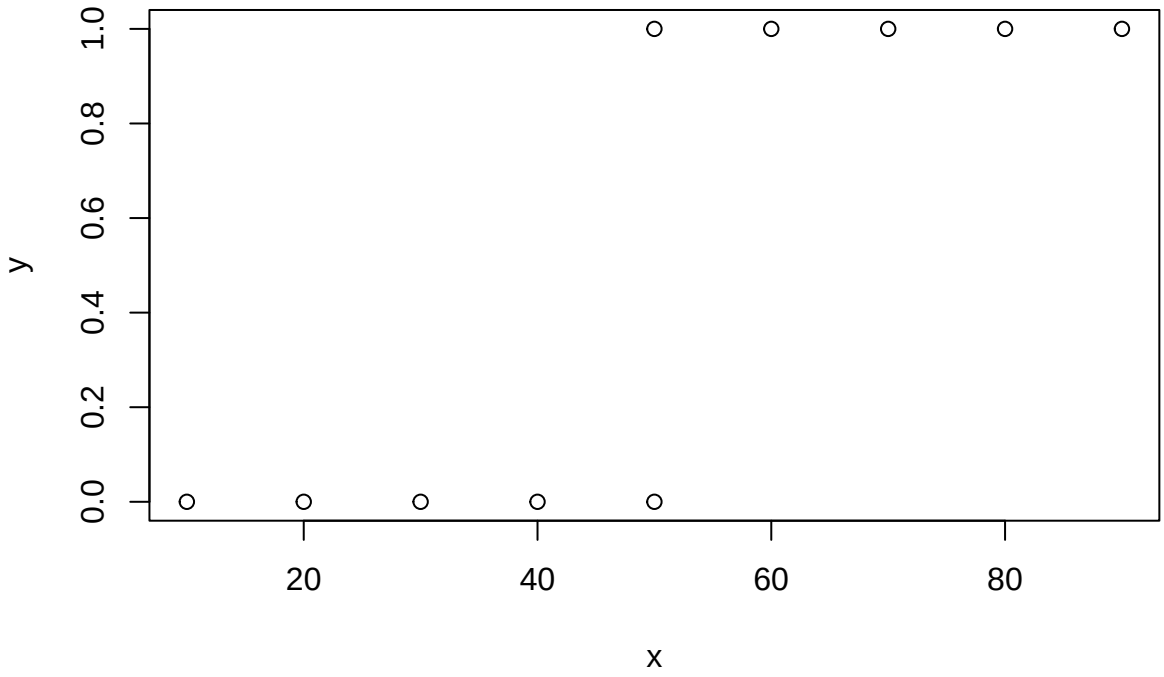


Figure 9: Logistic Regression Data for Quasi-Complete Separation Example of Agresti

13.2 Fit Model

```
lout <- llmdr(y ~ x, family = "binomial")
summary(lout)
```

```
##
## Call:
## llmdr(formula = y ~ x, family = "binomial")
##
##               Estimate       GDOR Std. Error z value Pr(>|z|)
```

```
## (Intercept)  4.710e-16 -1.250e+00  1.414e+00       0        1
## x                   NA  2.500e-02         NA      NA       NA
##
## Maximum likelihood distribution is the limit of distributions
## with parameter vectors Estimate + s * GDOR as s goes to infinity
## (NA in the Estimate column should be read as zero).
```

Now we see that the model is not completely degenerate (because the `Estimates` column of the printout is not all `NA`) but is partially degenerate (because the printout has a `GDOR` column). That's what Agresti calls quasi-complete separation.

As usual mean value parameters are easier to interpret.

```
data.frame(x, y, mu = estimates(lout, "mu"))
```

```
##     x y  mu
## 1  10 0 0.0
## 2  20 0 0.0
## 3  30 0 0.0
## 4  40 0 0.0
## 5  50 0 0.5
## 6  60 1 1.0
## 7  70 1 1.0
## 8  80 1 1.0
## 9  90 1 1.0
## 10 50 1 0.5
```

All of the components of the MLE $\hat{\mu}$ are on the boundary except those corresponding to $x = 50$ where we have one success and one failure and predict $\hat{\mu}_i = 0.5$.

13.3 Valid Region for Beta

```
# names here are odd, these are the model matrix and response vector
# for the data that are fixed at their observed values in the LCM
modmat.lcm <- as.matrix(lout$x[x != 50, ])
y.lcm <- y[x != 50]

logl <- logl.factory(modmat.lcm, y.lcm)

logl.gradient <- logl.gradient.factory(modmat.lcm, y.lcm)

level1 <- level2 <- sqrt(level)
level1
```

```
## [1] 0.9746794
```

```
alpha <- 1 - level2
crit <- (- log(alpha))
crit
```

```
## [1] 3.676138
```

```
beta.geyer <- do.curve(function(beta) crit + logl(beta), logl.gradient,
    cutoff = -1000)
```

```
foo <- beta.geyer[,1] > -8
plot(beta.geyer[,1], beta.geyer[,2],
    xlab = expression(beta[1]), ylab = expression(beta[2]),
```

```
    xlim = range(beta.geyer[foo,1]),
    ylim = range(beta.geyer[foo,2]), type = "n")
lines(beta.geyer[,1], beta.geyer[,2], col = "blue")
arrows(-5, 0.125, -5 + 2 * lout$gdor[1], 0.125 + 2 * lout$gdor[2])
```

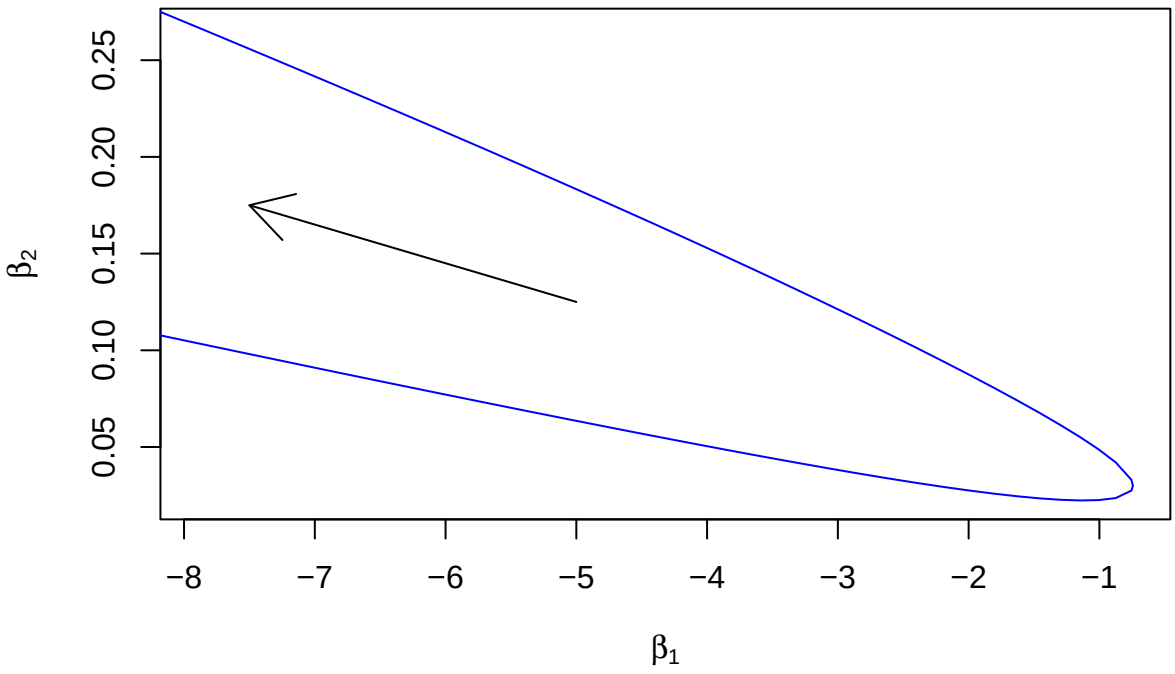


Figure 10: Confidence Region Consisting of β satisfyingeqrefeq:confreg-geyer. Arrow is GDOR

The blue curves in Figures 4 and 10 are different because the confidence levels are different.

Now we want to add the region of β satisfying (3). First we get the likelihood-based confidence region for the LCM. Here we can do that the easy way using R function `confint` from R package `MASS`.

```
gout <- glm(c(0, 1) ~ 1, family = "binomial")
summary(gout)
```

```
##
## Call:
## glm(formula = c(0, 1) ~ 1, family = "binomial")
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) 4.710e-16  1.414e+00       0        1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2.7726  on 1  degrees of freedom
## Residual deviance: 2.7726  on 1  degrees of freedom
```

```
## AIC: 4.7726
##
## Number of Fisher Scoring iterations: 2
```

```
cout <- confint(gout, level = level1)
```

```
## Waiting for profiling to be done...
```

```
cout
```

```
##     1.3 %    98.7 %
## -3.845904  3.845904
```

So now we know the likelihood-based confidence region for β_1 is $-3.85 < \beta_1 < 3.85$ when we constrain $\beta_2 = 0$. (R function `summary` says `NA` for coefficients it constrains to be zero.)

Now we know that the DOR for the OM is the only DOC for the LCM because we have only one non-identifiable parameter for the LCM (β_2) so only one DOC.

Thus our confidence region of β satisfying (3) is the set of all β of the form

$$(\beta_1, 0) + s\delta$$

with $-3.85 < \beta_1 < 3.85$ and s any real number and δ the GDOR. Its boundaries are the straight lines

$$(\pm 3.85, 0) + s\delta$$

So add them to our plot.

```
plot(beta.geyer[,1], beta.geyer[,2],
    xlab = expression(beta[1]), ylab = expression(beta[2]),
    xlim = range(beta.geyer[foo,1]),
    ylim = range(beta.geyer[foo,2]), type = "n")
lines(beta.geyer[,1], beta.geyer[,2], col = "blue")
arrows(-5, 0.125, -5 + 2 * lout$gdor[1], 0.125 + 2 * lout$gdor[2])

gdor <- lout$gdor
slope <- gdor[2] / gdor[1]
intercept.lower <- (- slope * cout[1])
intercept.upper <- (- slope * cout[2])
abline(intercept.lower, slope, col = "pink3")
abline(intercept.upper, slope, col = "pink3")
```

Our confidence region for β is the intersection of the two regions, the region inside the blue curve and between the region between the pink lines in Figure 11. The lower pink line does eventually intersect the blue curve, but that is off the plot to the left.

13.4 Intervals for Beta

So confidence intervals for β are

```
beta <- matrix(NA_real_, 2, 2)
colnames(beta) <- c("lower", "upper")
rownames(beta) <- names(gdor)
beta[1, 1] <- -Inf
beta[2, 2] <- Inf
beta[1, 2] <- max(beta.geyer[ , 1])
beta[2, 1] <- min(beta.geyer[ , 2])
beta
```

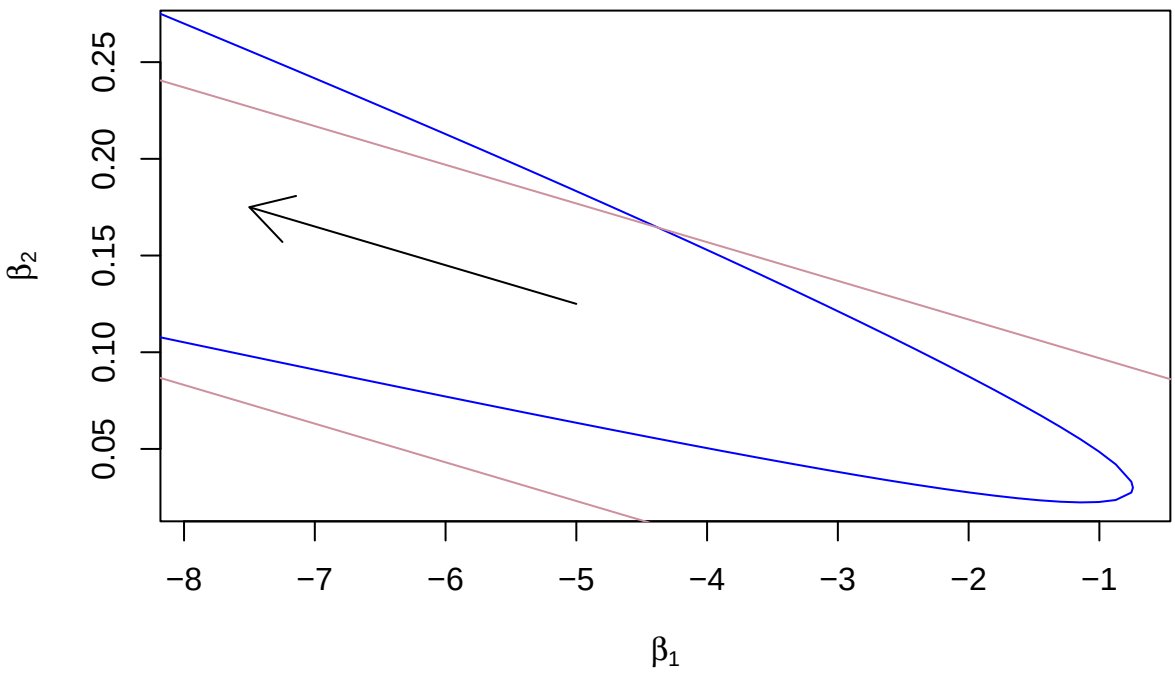


Figure 11: Confidence Regions. Blue: β satisfying eqrefeq:confreg-geyer. Pink: β satisfying eqrefeq:confreg-lcm. Arrow is GDOR

```
##                   lower      upper
## (Intercept)        -Inf -0.7432528
## x           0.02235808        Inf
```

We won't bother to make these intervals more precise using optimization. They are almost uninterpretable anyway. Clearly they present a lot less information than Figure 11.

13.5 Intervals for Theta

First we need to find the intersection of our two confidence regions.

```
nrow(beta.geyer)
```

```
## [1] 875
```

```
beta.upper <- beta.geyer[ , 1] * slope + intercept.upper
is.up <- beta.geyer[ , 2] > beta.upper
range(which(is.up))
```

```
## [1] 457 875
```

```
beta.lower <- beta.geyer[ , 1] * slope + intercept.lower
is.dn <- beta.geyer[ , 2] < beta.lower
range(which(is.dn))
```

```
## [1]   1 399
```

```
beta.both <- beta.geyer
beta.both[is.up, 2] <- beta.geyer[is.up, 1] * slope + intercept.upper
beta.both[is.dn, 2] <- beta.geyer[is.dn, 1] * slope + intercept.lower
```

Then we map to θ

```
modmat <- as.matrix(lout$x)
theta.both <- tcrossprod(modmat, beta.both)
theta <- apply(theta.both, 1, range)
theta <- t(theta)
colnames(theta) <- c("lower", "upper")
theta
```

```
##           lower        upper
## 1  -801.7587785  -0.42512173
## 2  -602.2805598  -0.03051594
## 3  -402.8023410   0.64116247
## 4  -203.3241223   2.18817937
## 5    -3.8459036   3.84590355
## 6    -2.1787839 204.82715993
## 7    -0.6411474 405.80841631
## 8     0.0293836 606.78967268
## 9     0.4235842 807.77092906
## 10   -3.8459036   3.84590355
```

```
# fix up infinite by hand
theta[x < 50, "lower"] <- -Inf
theta[x > 50, "upper"] <- Inf
theta
```

```
##         lower       upper
## 1        -Inf -0.42512173
```

```
## 2        -Inf -0.03051594
## 3        -Inf  0.64116247
## 4        -Inf  2.18817937
## 5  -3.8459036  3.84590355
## 6  -2.1787839         Inf
## 7  -0.6411474         Inf
## 8   0.0293836         Inf
## 9   0.4235842         Inf
## 10 -3.8459036  3.84590355
```

We might as well polish these up with optimization because we need the R function `do.theta.intervals` defined in Section 10.9 above modified to do this problem in order to do the rest of our analysis of this example.

```
# names here are odd, these are the model matrix and response vector
# for the data that are free in the LCM
modmat.lcm <- as.matrix(lout$x[lout$lcm.support, ])
y.lcm <- y[lout$lcm.support]
modmat.lcm
```

```
##    (Intercept)  x
## 5            1 50
## 10           1 50
```

```
y.lcm
```

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

```
logl.lcm <- logl.factory(modmat.lcm, y.lcm)

logl.lcm.gradient <- logl.gradient.factory(modmat.lcm, y.lcm)

beta.hat.lcm <- c(0, 0)

crit2 <- crit
crit1 <- qchisq(level1, df = sum(! is.na(lout$coefficients))) / 2
crit1
```

```
## [1] 2.500914
```

```
crit2
```

```
## [1] 3.676138
```

```
# now constraint function for optimization has both constraints

hin <- function(beta) c(logl(beta) + crit2,
    logl.lcm(beta) - logl.lcm(beta.hat.lcm) + crit1)

hin.jac <- function(beta)
    rbind(logl.gradient(beta), - logl.lcm.gradient(beta))

i <- sample(which(is.up), 1)
beta.both[i, ]; hin(beta.both[i, ]); hin.jac(beta.both[i, ])
```

```
## [1] -939.40964   18.86511
```

```
## [1]  3.6761383471 -0.0009793215
```

```
##        (Intercept)             x
```

```
## [1,] -5.494406e-81 -2.197260e-79
## [2,]  9.581598e-01  4.790799e+01
```

```
jacobian(hin, beta.both[i, ])
```

```
##            [,1]      [,2]
## [1,]  0.0000000   0.00000
## [2,] -0.9581598 -47.90799
```

```
a.little.up <- c(0, 1e-4)
a.little.dn <- c(0, -1e-4)
hin(beta.both[i, ] + a.little.up); hin(beta.both[i, ] + a.little.dn)
```

```
## [1]  3.676138347 -0.005770632
```

```
## [1] 3.676138347 0.003810965
```

It seems a bit odd that the constraint values returned by R function `hin` are neither zero and the closest is -9.793215×10^{-4}, but that is what inexact computer arithmetic and the default convergence tolerances for R function `auglag` do.

If we take that number to correspond to a zero using infinite precision arithmetic, then this is what we expect. We are on the pink boundary where the second constraint is zero. Oh. The inexactness here might be where the pink lines are drawn which might be the fault of R function `confint.glm` in R package `MASS`. We won't sort out who is to blame, because we will now be using R function `hin` so say when we are in the confidence region for β. We won't use the pink lines any more.

The last two lines show that if we go a little up, then we are outside the confidence region (one component of constraint function negative), whereas if we go a little down, then we are inside (both components positive).

```
i <- sample(which(is.dn), 1)
beta.both[i, ]; hin(beta.both[i, ]); hin.jac(beta.both[i, ])
```

```
## [1] -83.93439   1.60177
```

```
## [1]  3.6761331704 -0.0009793215
```

```
##        (Intercept)             x
## [1,]  5.171976e-06  3.103658e-04
## [2,] -9.581598e-01 -4.790799e+01
```

```
jacobian(hin, beta.both[i, ])
```

```
##              [,1]         [,2]
## [1,] 5.171976e-06 3.103658e-04
## [2,] 9.581598e-01 4.790799e+01
```

```
hin(beta.both[i, ] + a.little.up); hin(beta.both[i, ] + a.little.dn)
```

```
## [1] 3.676133201 0.003810965
```

```
## [1]  3.676133139 -0.005770632
```

Again, looks OK, except for inexactness of something or other. Now a little up is inside and a little down is outside.

```
i <- sample(which((! is.dn) & (! is.up)), 1)
beta.both[i, ]; hin(beta.both[i, ]); hin.jac(beta.both[i, ])
```

```
## [1] -4.1528532  0.1576143
```

```
## [1] 5.955298e-10 1.118252e-01
```

```
##      (Intercept)         x
## [1,]  -1.8691306 -60.72811
## [2,]   0.9530408  47.65204
```

```
jacobian(hin, beta.both[i, ])
```

```
##            [,1]      [,2]
## [1,] -1.8691306 -60.72811
## [2,] -0.9530408 -47.65204
```

Now this is a point on the blue curve, and here the first constraint is nearly zero, and the jacobian looks OK here too. So it looks like our constraint functions and their gradients are OK.

```
do.theta.intervals <- function(x) {
    foo <- cbind(1, x) %*% t(beta.both)
    gdor <- lout$gdor

    lower <- rep(-Inf, length(x))
    upper <- rep(+Inf, length(x))

    zapsmall <- function(x) if (abs(x) < sqrt(.Machine$double.eps)) 0 else x

    for (i in seq(along = x)) {
        eta <- sum(c(1, x[i]) * gdor)
        eta <- zapsmall(eta)
        if (eta <= 0) {
            # need upper bound
            idx <- which(foo[i, ] == max(foo[i, ]))
            idx <- idx[1] # just in case of ties
            beta.start <- beta.both[idx, ]
            aout <- auglag(beta.start,
                fn = function(beta) sum(c(1, x[i]) * beta),
                gr = function(beta) c(1, x[i]),
                hin = hin,
                hin.jac = hin.jac,
                control.outer = list(trace = FALSE),
                control.optim = list(fnscale = -1))
            # looks like we have to accept 9, which is
            # Convergence due to lack of progress in parameter updates
            stopifnot(aout$convergence %in% c(0, 9))
            upper[i] <- aout$value
        }
        if (eta >= 0) {
            # need lower bound
            idx <- which(foo[i, ] == min(foo[i, ]))
            idx <- idx[1] # just in case of ties
            beta.start <- beta.both[idx, ]
            aout <- auglag(beta.start,
                fn = function(beta) sum(c(1, x[i]) * beta),
                gr = function(beta) c(1, x[i]),
                hin = hin,
                hin.jac = hin.jac,
                control.outer = list(trace = FALSE))
            # looks like we have to accept 9, which is
            # Convergence due to lack of progress in parameter updates
            stopifnot(aout$convergence %in% c(0, 9))
```

```
            lower[i] <- aout$value
        }
    }
    cbind(lower, upper)
}
```

```
theta <- do.theta.intervals(sort(unique(x)))
rownames(theta) <- sort(unique(x))
theta
```

```
##          lower       upper
## 10        -Inf -0.42316007
## 20        -Inf -0.02918282
## 30        -Inf  0.64116395
## 40        -Inf  2.18817937
## 50 -3.84590355  3.84590355
## 60 -2.20143915         Inf
## 70 -0.64116375         Inf
## 80  0.02918285         Inf
## 90  0.42315998         Inf
```

13.6 Intervals for Mu

And we can map these to the mean value parameter.

```
mu <- 1 / (1 + exp(- theta))
mu
```

```
##         lower     upper
## 10 0.00000000 0.3957608
## 20 0.00000000 0.4927048
## 30 0.00000000 0.6550165
## 40 0.00000000 0.8991830
## 50 0.02092009 0.9790799
## 60 0.09962133 1.0000000
## 70 0.34498352 1.0000000
## 80 0.50729519 1.0000000
## 90 0.60423916 1.0000000
```

And plot them.

```
errbar(sort(unique(x)), as.numeric(sort(unique(x)) > 50), mu[ , 2], mu[ , 1],
    xlab = "x", ylab = "probability")
# add the other data point for x = 50
points(50, 1)
```

13.7 Submodel Canonical Statistic

This is like Section 10.10 except for the difference in the data.

```
yy <- matrix(0:1, nrow = 2, ncol = length(x))
colnames(yy) <- paste0("y", seq(along = x))
yy <- expand.grid(as.data.frame(yy))

head(yy)
```

```
##   y1 y2 y3 y4 y5 y6 y7 y8 y9 y10
```

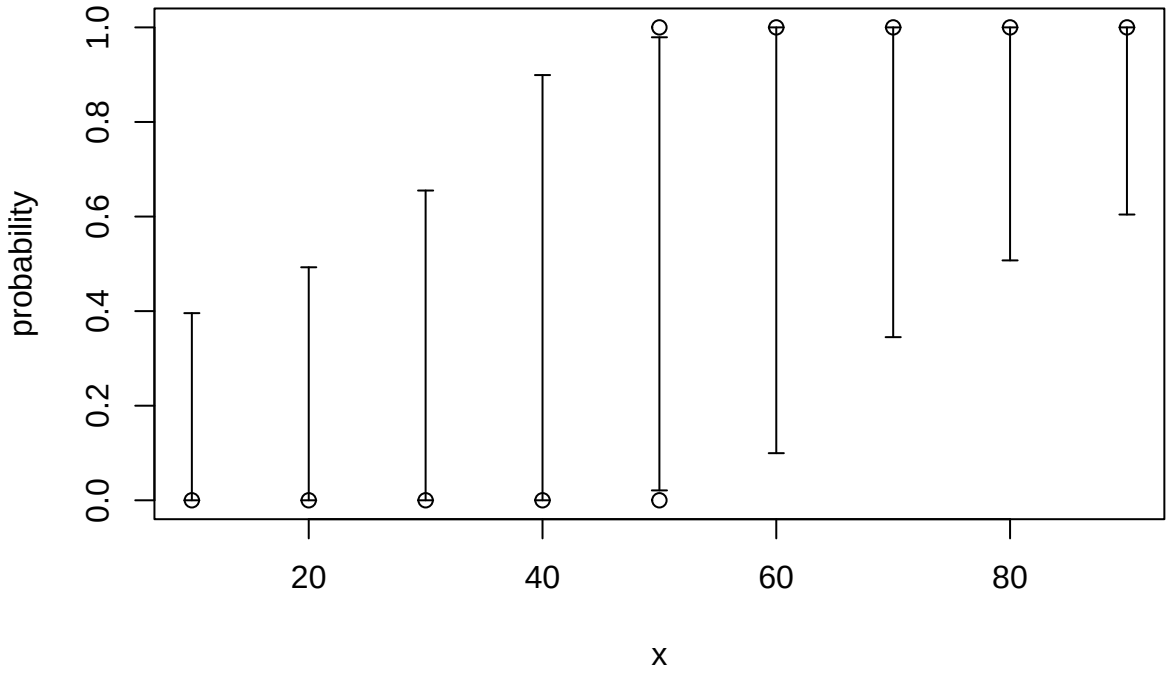


Figure 12: Confidence intervals for Mean Value Parameters. Hollow circles are MLE.

```
## 1  0  0  0  0  0  0  0  0  0   0
## 2  1  0  0  0  0  0  0  0  0   0
## 3  0  1  0  0  0  0  0  0  0   0
## 4  1  1  0  0  0  0  0  0  0   0
## 5  0  0  1  0  0  0  0  0  0   0
## 6  1  0  1  0  0  0  0  0  0   0
```

```
dim(yy)
```

```
## [1] 1024   10
```

Get distinct possible values of the submodel canonical statistic and the observed values.

```
m <- cbind(1, x)
mtyy <- t(m) %*% t(yy)
t1 <- mtyy[1, ]
t2 <- mtyy[2, ]
t1.obs <- sum(y)
t2.obs <- sum(x * y)
```

And make the plot.

```
t.key <- paste(t1, t2, sep = ":")
t.idx <- match(unique(t.key), t.key)
t1.uniq <- t1[t.idx]
t2.uniq <- t2[t.idx]

par(mar = c(5, 5, 1, 0) + 0.1)
plot(t1.uniq, t2.uniq, xlab = expression(t[1] == sum(y)),
    ylab = expression(t[2] == sum(x * y)))
points(t1.obs, t2.obs, pch = 19)

# add H
gdor <- lout$gdor
# rotate 90 degrees to convert normal vector to tangent vector
tanv <- rev(gdor) * c(-1, 1)
tanv
```

```
##           x (Intercept)
##      -0.025      -1.250
```

```
# now convert to slope and intercept of line
slope <- tanv[2] / tanv[1]
intercept <- t2.obs - t1.obs * slope
abline(intercept, slope)
```

This differs from Figure 6 in that now the support of the LCM, the points that lie on the hyperplane H, which is the line in the figure, has three points rather than one. Hence the LCM is not completely degenerate (concentrated at one point) but rather only *partially degenerate*.

But otherwise the picture is much the same. The MLE does not exist in the OM because the observed data (for the submodel canonical sufficient statistic) is on the boundary of the convex hull of the set of its possible values.

13.8 Region for Tau

This is like Section 10.12 except for the difference in data.

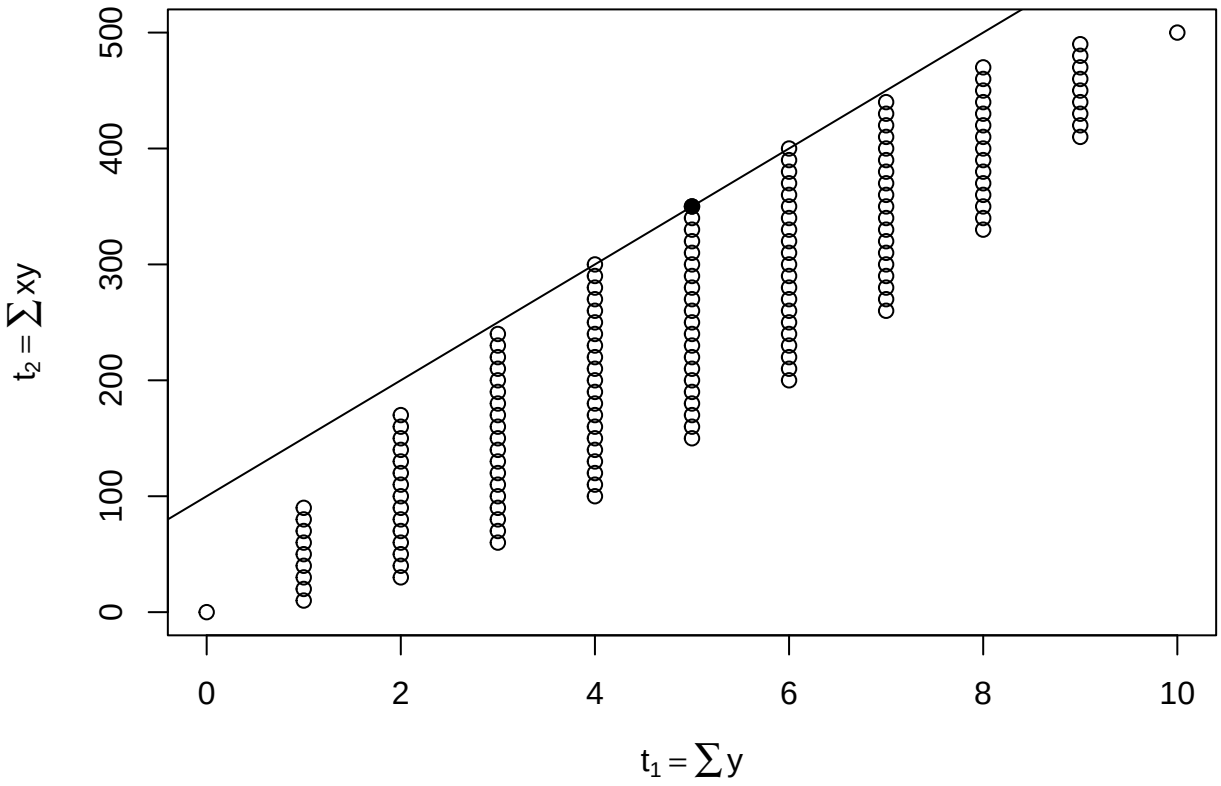
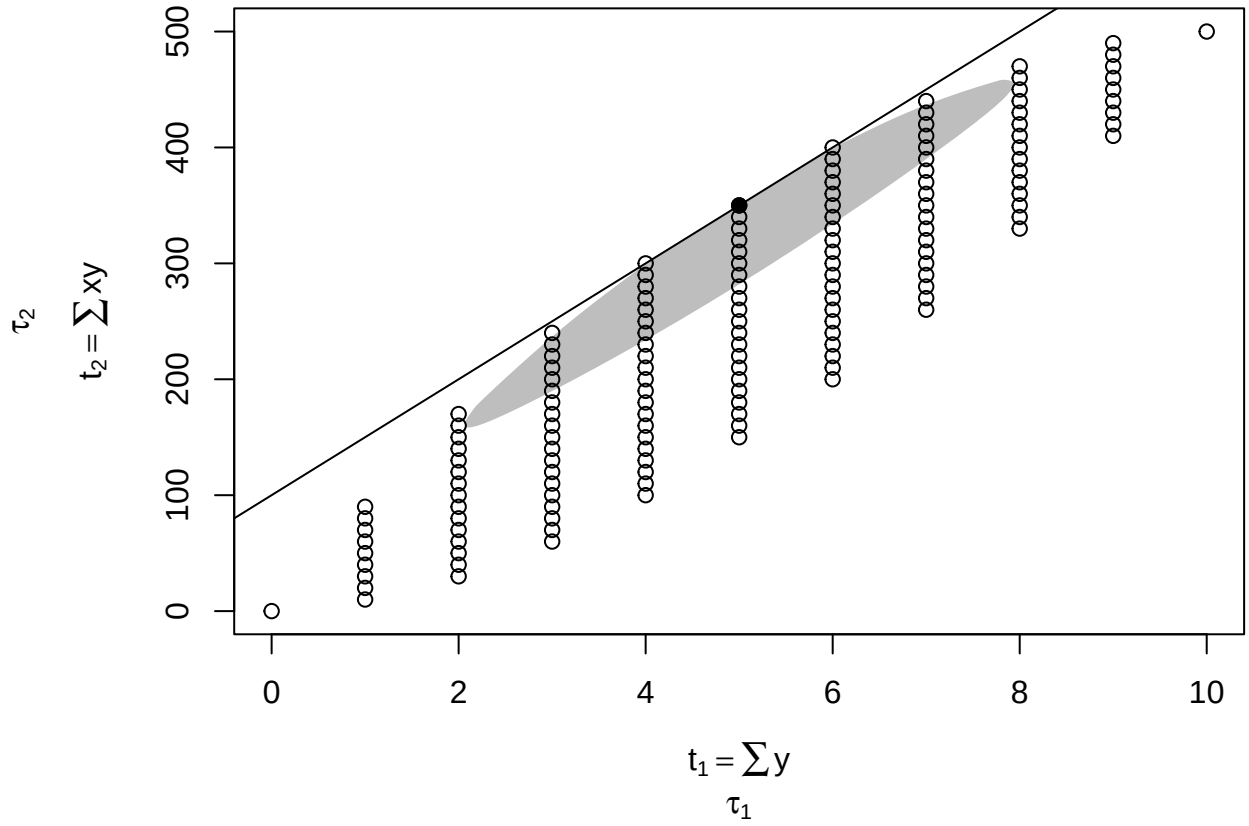


Figure 13: Possible values of the submodel canonical statistic vector $M^T y$ for the data shown in Figure reffig:agresti-quasi-data-plot. Solid dot is the observed value. Line is hyperplane eqrefeq:h determined by the GDOR.

```
theta <- tcrossprod(modmat, beta.both)
mu <- 1 / (1 + exp(- theta))
tau <- crossprod(modmat, mu)
# add observed value to these points
tau <- cbind(c(t1.obs, t2.obs), tau)

par(mar = c(5, 6, 1, 0) + 0.1)
plot(t1.uniq, t2.uniq, xlab = expression(t[1] == sum(y)),
    ylab = expression(t[2] == sum(x * y)), type = "n")
polygon(tau[1, ], tau[2, ], border = NA, col = "gray")
points(t1.uniq, t2.uniq)
points(t1.obs, t2.obs, pch = 19)
abline(intercept, slope)
mtext(expression(tau[1]), side = 1, line = 4)
mtext(expression(tau[2]), side = 2, line = 5)
```

\begin{figure}



{

}

\caption{Same as Figure 13 Except Gray Region is 95% Confidence Region for Submodel Mean Value Parameter} \end{figure}

13.9 Quasi-Complete Separation

This is like Section 10.11 except for the difference in data.

```
dim(modmat)
```

```
## [1] 10  2
```

```
plot(modmat[ , 1], modmat[ , 2],
    xlab = colnames(modmat)[1], ylab = colnames(modmat)[2])
intercept <- 50 - 1 * slope
abline(intercept, slope)
points(modmat[y == 1, 1], modmat[y == 1, 2], pch = 19)
points(modmat[x == 50, 1], modmat[x == 50, 2], pch = 19, col = "gray")
```

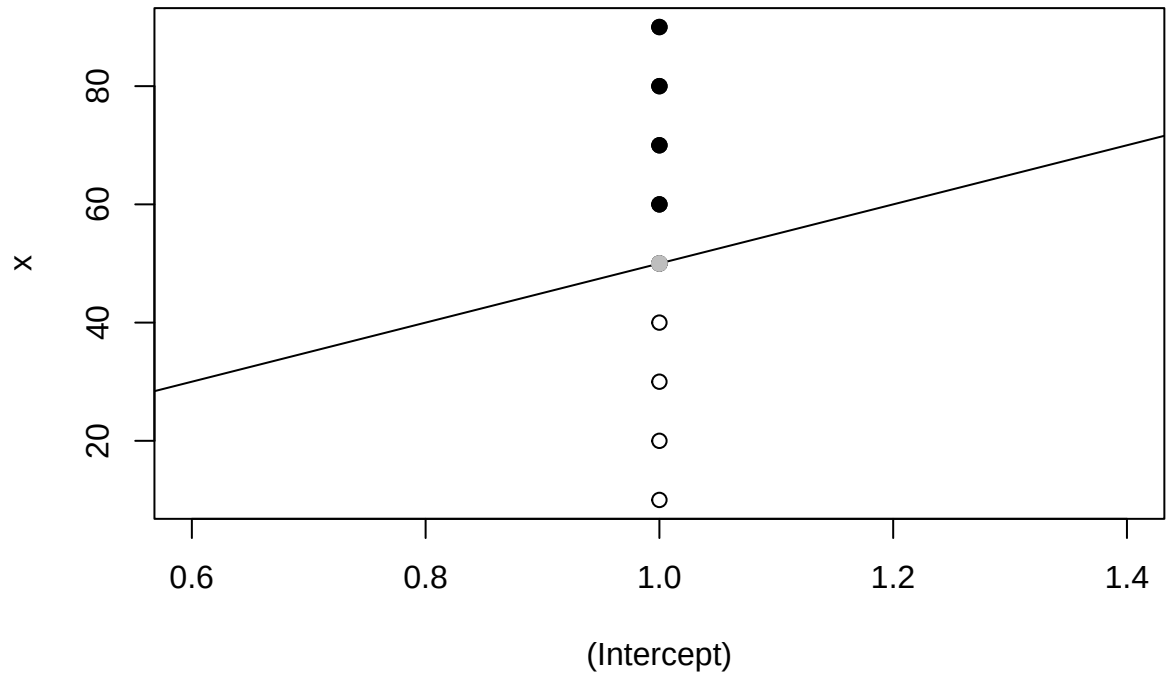


Figure 14: Quasi-Complete Separation of Successes and Failures. Black dots successes. White dots failures. Gray dot, one success and one failure. Black line separating hyperplane

13.10 Confidence Intervals for Hypothetical Individuals

This is like Section 10.14 except for the difference in data.

```
xx <- seq(0, 100, 0.1)
# be especially careful at jump
xx <- c(xx, 49.99, 50.01)
xx <- sort(xx)
theta <- do.theta.intervals(xx)
mu <- 1 / (1 + exp(- theta))
# have to do by hand since optimization isn't good enough
mu[xx < 50, 1] <- 0
```

```
mu[xx > 50, 2] <- 1
```

```
plot(x, y, type = "n",
    xlab = expression(x), ylab = expression(mu(x)))
xxx <- c(xx, rev(xx))
yyy <- c(mu[ , 1], rev(mu[ , 2]))
length(xxx); length(yyy)
```

```
## [1] 2006
```

```
## [1] 2006
```

```
polygon(xxx, yyy, border = NA, col = "gray")
points(x, y)
```

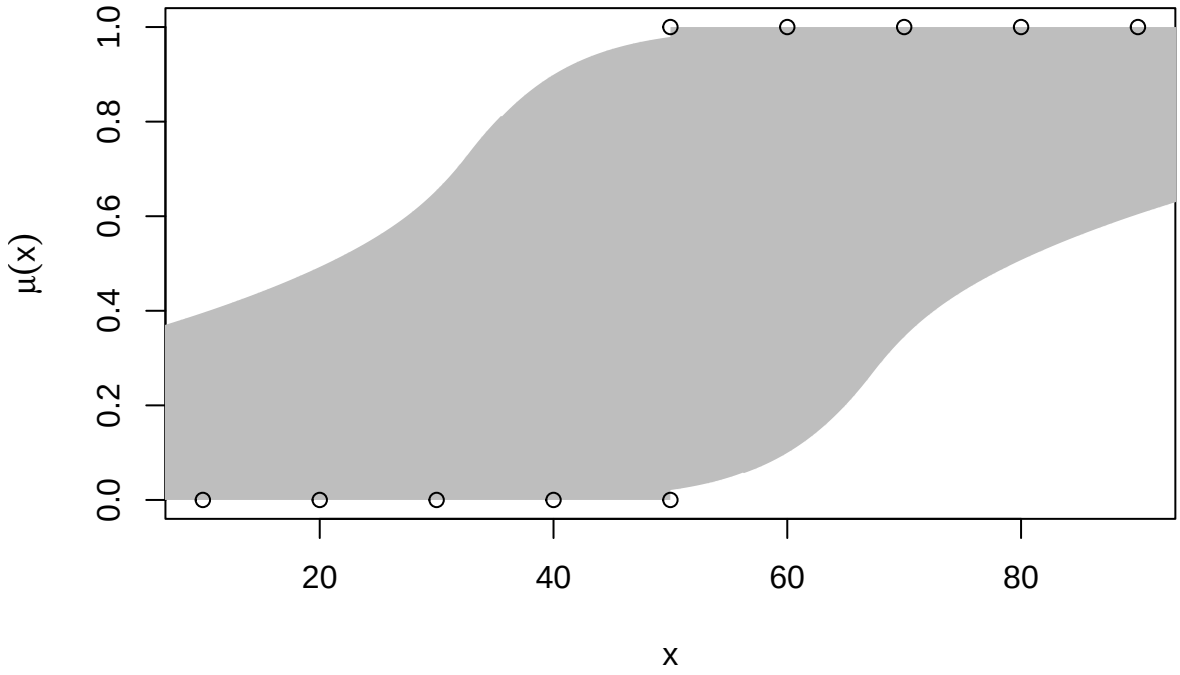


Figure 15: Like Figurereffig:agresti-quasi-mu-plot Except with Confidence Intervals for Every x value

That finishes our work on this toy problem.

14 Theory of Hypothesis Tests

The theory of hypothesis tests propounded in Geyer (2009) is very simple, once you understand it. Geyer (2009) attributes it to Stephen Fienberg (who outlined this theory in answer to a question from your humble author asked in the questions after a talk Fienberg was giving).

The main point is that in the theory of hypothesis testing, the distribution of the data used for calculations is the distribution *under the null hypothesis*. (This ignores power calculations.) Thus the LCM for the

alternative hypothesis is irrelevant. (And it would be irrelevant even if we were doing power calculations, because those only involve alternatives near the null hypothesis.)

The next most important point is that the MLE distribution *under the null hypothesis* is a conditional distribution (if you include conditioning on nothing, which is the same as not conditioning, in case no GDOR exists and the MLE is in the OM). Thus we have to fit the null and alternative hypotheses *using the same conditioning* for each.

Everything else is straightforward. It is just the usual theory of hypothesis tests associated with maximum likelihood estimation (Wilks, Rao, Wald) applied to this conditional model. Actually, since Wald tests only use the MLE for the alternative hypothesis, they would only tell us about LCM for the alternative, so they don't make much sense except in the output of R function `summary`.

It follows that we do not even need the MLE calculated by R function `llmdr` for the *alternative hypothesis* (for Wilks or Rao). But we can use the deviance it calculates.

It follows that we have to refit the *alternative hypothesis* using the conditioning for the *null hypothesis*. The method of R generic function `anova` for objects of class `llmdr` does this. The appropriate degrees of freedom for the *alternative hypothesis* will be the degrees of freedom of this refitted model. And these degrees of freedom will not be either of the degrees of freedom in the object returned by R function `llmdr` when it calculates the MLE for the *alternative hypothesis*.

Thus a model has many possible degrees of freedom

- One is the number of identifiable parameters of the OM. This is the degrees of freedom one should use for AIC and BIC (maybe, for a counterargument, see the section about this below).
- Another is the number of identifiable parameters of the LCM.
- Yet another is the number of identifiable parameters of the model when considered as an alternative hypothesis. This depends on what the null hypothesis is. In general, it can be anything between the two degrees of freedom listed above.

In summary, our theory of hypothesis testing is fairly conventional. The test statistic (Wilks, Rao, or Wald) still has an asymptotic chi-square distribution. But, unless the MLE for the null hypothesis in in its OM, the *degrees of freedom are different* from conventional.

So we just have to remember that the degrees of freedom for the alternative hypothesis are different from conventional and depend on what the null hypothesis is.

15 Categorical Data Example of Geyer

This is the example in Section 2.3 of Geyer (2009). It is a seven-dimensional contingency table, each dimension having two values, so the number of cells of the table is $2^7 = 128$. These data are found in dataset `catrec` in R package `llmdr`

```
data(catrec)
class(catrec)
```

```
## [1] "data.frame"
```

```
dim(catrec)
```

```
## [1] 128   8
```

```
names(catrec)
```

```
## [1] "v1" "v2" "v3" "v4" "v5" "v6" "v7" "y"
```

The response (vector of cell counts) is `y`. The other variables are the indicators for each dimension.

Geyer (2009) fits the model with all possible three-way interactions and no higher interactions

```
lout <- llmdr(y ~ (v1 + v2 + v3 + v4 + v5 + v6 + v7)^3,
    family = "poisson", data = catrec, proofs = 2)
length(lout$coefficients)
```

```
## [1] 64
```

```
sum(is.na(lout$coefficients))
```

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

We don't print out the summary, because it is too long and is about meaningless parameters anyway.

We notice that the dimension of the LCM is one less than the dimension of the OM, which means the GDOR points in a unique direction (every GDOR for this problem points in the same direction).

This GDOR has nonnegative components

```
# did proofs = 2 above to get GDOR with exactly correct zeros
cbind(lout$gdor[lout$gdor != 0])
```

```
##             [,1]
## (Intercept)   -1
## v1             1
## v2             1
## v3             1
## v5             1
## v1:v2         -1
## v1:v3         -1
## v1:v5         -1
## v2:v3         -1
## v2:v5         -1
## v3:v5         -1
## v1:v2:v3       1
## v1:v3:v5       1
## v2:v3:v5       1
```

This agrees with Table 1 in Geyer (2009).

Geyer (2009) considers the following hypothesis tests.

```
lout0 <- llmdr(y ~ (v1 + v2 + v3 + v4 + v5 + v6 + v7)^2,
    family = "poisson", data = catrec)
lout2 <- llmdr(y ~ (v1 + v2 + v3 + v4 + v5 + v6 + v7)^4,
    family = "poisson", data = catrec)
anova(lout0, lout, lout2)
```

```
## Analysis of Deviance Table
##
## Model 1: y ~ (v1 + v2 + v3 + v4 + v5 + v6 + v7)^2
## Model 2: y ~ (v1 + v2 + v3 + v4 + v5 + v6 + v7)^3
## Model 3: y ~ (v1 + v2 + v3 + v4 + v5 + v6 + v7)^4
##   Df as Null Df as Alt Deviance Df for test     LRT P-value
## 1         29         0  191.629
## 2         63        64   31.291          35 160.338 0.00000
## 3         94        94   16.067          31  15.224 0.99212
```

This says the model with all three-way interactions and no higher-order interactions, which has OM $\neq$ LCM, fits the data much better than the model with only two-way interactions. And this model fits just as well as the model with all four-way interactions.

Note that in line two of the printout, the degrees of freedom are different for the three-way interactions model, depending on whether is has the role of null hypothesis or alternative hypothesis.

There are, of course, many, many more models to consider. But, as this is being written, R function `glmbb` in R package `glmbb` (Geyer, 2020) uses up all memory the computer has and then crashes when applied to this problem, and R functions `add1`, `drop1`, and `step` do have not have methods for objects returned by R function `llmdr`.

So we could do whatever hypothesis tests we can think of, but we will have to wait until R package `llmdr` is more complete or R package `glmbb` is fixed to use less memory in order to search among the very many models for these data.

16 Multinomial and Product Multinomial Sampling

Unlike R function `glm` the new function `llmdr` also does multinomial and product multinomial sampling. Let's see how that works.

This is a competitor for "response" in scare quotes, which we discussed in the notes accompanying Agresti, Chapter 8.

16.1 Alligators

Let's redo an example previously done in those notes.

```
data(table_8.1)
alligator <- transform(table_8.1,
    lake = factor(lake,
        labels = c("Hancock", "Oklawaha", "Trafford", "George")),
    gender = factor(gender, labels = c("Male", "Female")),
    size = factor(size, labels = c("<=2.3", ">2.3")),
    food = factor(food,
        labels = c("Fish", "Invertebrate", "Reptile", "Bird", "Other")))
names(alligator)
```

```
## [1] "lake"   "gender" "size"   "food"   "count"
```

```
lout <- llmdr(count ~ food * (size + gender + lake),
    strata = ~ size * lake * gender,
    family = "multinomial", data = alligator)
summary(lout)
```

```
##
## Call:
## llmdr(formula = count ~ food * (size + gender + lake), family = "multinomial",
##     data = alligator, strata = ~size * lake * gender)
##
##                                Estimate Std. Error z value Pr(>|z|)
## foodInvertebrate               -2.0745     0.6117  -3.392 0.000695 ***
## foodReptile                    -2.9141     0.8856  -3.290 0.001000 **
## foodBird                       -2.4633     0.7739  -3.183 0.001458 **
## foodOther                      -0.9167     0.4782  -1.917 0.055217 .
## foodInvertebrate:size>2.3      -1.3363     0.4112  -3.250 0.001155 **
## foodReptile:size>2.3            0.5570     0.6466   0.861 0.388977
## foodBird:size>2.3               0.7302     0.6523   1.120 0.262919
## foodOther:size>2.3             -0.2906     0.4599  -0.632 0.527515
## foodInvertebrate:genderFemale   0.4630     0.3955   1.171 0.241796
```

```
## foodReptile:genderFemale        0.6276     0.6853   0.916 0.359785
## foodBird:genderFemale           0.6064     0.6888   0.880 0.378668
## foodOther:genderFemale          0.2526     0.4663   0.542 0.588100
## foodInvertebrate:lakeOklawaha   2.6937     0.6693   4.025 5.70e-05 ***
## foodReptile:lakeOklawaha        1.4008     0.8105   1.728 0.083926 .
## foodBird:lakeOklawaha          -1.1256     1.1924  -0.944 0.345159
## foodOther:lakeOklawaha         -0.7405     0.7422  -0.998 0.318377
## foodInvertebrate:lakeTrafford   2.9363     0.6874   4.272 1.94e-05 ***
## foodReptile:lakeTrafford        1.9316     0.8253   2.340 0.019263 *
## foodBird:lakeTrafford           0.6617     0.8461   0.782 0.434146
## foodOther:lakeTrafford          0.7912     0.5879   1.346 0.178400
## foodInvertebrate:lakeGeorge     1.7805     0.6232   2.857 0.004277 **
## foodReptile:lakeGeorge         -1.1295     1.1928  -0.947 0.343689
## foodBird:lakeGeorge            -0.5753     0.7952  -0.723 0.469429
## foodOther:lakeGeorge           -0.7666     0.5686  -1.348 0.177563
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This is the Right Thing. It does not show any coefficients for regressors in `strata` but does condition on them. The estimates and their standard errors are those for product multinomial sampling.

But this doesn't have anything to do with the subject of these notes except that we have proved that the MLE exists in the OM (something R function `glm` cannot do).

But when we ran R function `glmbb` on this, we had warnings.

```
gout <- glmbb(big = count ~ lake * gender * size * food,
    little = ~ lake * gender * size + food,
    family = poisson, data = alligator)
```

```
## Warning: glm.fit: fitted rates numerically 0 occurred
```

```
## Warning: glm.fit: fitted rates numerically 0 occurred
```

What were the models for which it should have given warnings?

```
foo <-gout$envir
rm(min.crit, envir = foo)
bar <- eapply(foo, function(x) x$formula)
lout <- lapply(bar, function(x) llmdr(x, family = "poisson", data = alligator))
is.lcm <- sapply(lout, function(x) x$is.lcm)
sum(is.lcm)
```

```
## [1] 9
```

(we use `family = "poisson"` here to match the formulas `glmbb` uses, and then we don't need argument `strata`).

R function `glm` should have given 9 warnings rather than two! So this is proof that its idea of warnings is completely inadequate. It gives lots of false negatives. It can also give false positives, but we haven't shown an example of that yet.

Here are the models with LCM

```
sapply(bar[is.lcm], tidy.formula.hierarchical)
```

```
##                                      sha1.e4402d800c21a9abb0762161d5e1c6e35e4ebde7
## "count ~ lake*gender*size + lake*gender*food + lake*size*food + gender*size*food"
##                                      sha1.fff51edf16f5ff2e1b31d3ce343bc7009bf43bba
##                          "count ~ gender*food + lake*gender*size + lake*size*food"
```

```
##                                     sha1.853e5b67c6699b1672c371fe92acc7892e98ec9e
##                       "count ~ lake*gender*size + lake*size*food + gender*size*food"
##                                     sha1.ae5b3169d85068f271a8d6c4d7d381d889be7c41
##                     "count ~ lake*gender*size + lake*gender*food + gender*size*food"
##                                     sha1.b3b8ef68c4353ba55356e3c8e41ec64317f43fac
##                       "count ~ lake*gender*size + lake*gender*food + lake*size*food"
##                                     sha1.9324271d9e5c822e27dffeb511f9eea66344692c
##                            "count ~ size*food + lake*gender*size + lake*gender*food"
##                                     sha1.e9b7454a669e0deeda786851802e9e69b2daf4a8
##                                       "count ~ lake*gender*size + lake*size*food"
##                                     sha1.858c44481b5873e02770b9e6c003b7246ecd25ab
##                                     "count ~ lake*gender*size + lake*gender*food"
##                                     sha1.fa8295479e78aa560f7648cbed1ab34bcb7511cc
##                                                  "count ~ lake*gender*size*food"
```

Eventually we need to make R package `glmbb` use R function `llmdr` to fit models rather than R function `glm`. But that is not done yet.

16.2 A Digression about AIC and BIC and LCM

16.2.1 AIC

To understand the issues here we need to do a little bit of theory. The mathematical argument that leads to AIC assumes the "usual asymptotics of maximum likelihood". MLE at infinity (for canonical parameters, on the boundary for mean value parameters) mean those assumptions are wrong.

On the other hand, those "usual asymptotics of maximum likelihood" may apply to the LCM if the conditioning that produces the LCM leaves enough components of the response vector random so the sample size can be considered "large" so the parameter estimates in the LCM are approximately normal. When the LCM is completely degenerate, so it has no parameters to estimate, the asymptotic argument does still apply if we consider the chi-square distribution on zero degrees of freedom to be the distribution of the constant random variable always equal to zero (which R does)

```
pchisq(0, df = 0)
```

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

```
pchisq(1e-16, df = 0)
```

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

It follows that the correct p to use in the definition

$$\text{AIC} = \text{deviance} + 2p$$

is what R function `llmdr` calls the degrees of freedom for the LCM, component `df` of the result it returns.

There really cannot be any argument about this if you follow the theory. AIC is trying to be an unbiased estimate of Kullback-Leibler information (times two). On or near the boundary of the mean value parameter space the Kullback-Leibler information function is flat or nearly flat in the same directions as the log likelihood function; there are only `df` dimensions in which it is strongly curved and can be overestimated.

For an explicit example where this is clearly correct, let us redo the complete separation example of Agresti just doing AIC.

Recall this analysis of deviance table

```
save.complete$anova
```

```
## Analysis of Deviance Table
##
## Model 1: y ~ 1
## Model 2: y ~ x
##   Df as Null Df as Alt Deviance Df for test   LRT    P-value
## 1          1         0    11.09
## 2          0         2     0.00           1 11.09 0.00086778
```

Now we want AIC. We claim that the correct AIC are

```
with(save.complete$null, deviance + 2 * df)
```

```
## [1] 13.09035
```

```
with(save.complete$alternative, deviance + 2 * df)
```

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

This is not what R function `glm` and friends (methods for generic functions that handle objects of class `"glm"`) calculates

```
with(save.complete, AIC(glm(null$formula, family = "binomial", data = data)))
```

```
## [1] 13.09035
```

```
with(save.complete, AIC(glm(alternative$formula, family = "binomial", data = data)))
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## [1] 4
```

Of course either calculation says the model with formula y ~ x is best, but that's not the point. The point is that we have issues with how to calculate AIC. This section recommends one thing. R function `glm` and friends do another.

There is no question that R function `glm` and friends are theoretically incorrect. The AIC argument says that it is a bias-corrected estimate of Kullback-Leibler information (times two). It corrects for overfitting. But the model with formula y ~ x cannot overfit. All randomness is gone in a completely degenerate LCM.

But I can imagine the counterarguments. Those arguments ignore the math behind AIC and only pay attention to the way it is usually used. Sparsity fans will argue that we are really looking for sparsity and AIC is a penalty that rewards sparsity, and if we redefine AIC the way we recommend here, that won't any longer be rewarding sparsity. And that was what was always good about AIC (never mind what theory says).

So I understand that a lot of people might not like what this section says (mathematically correct or no). They will continue to use their grandfather's version of AIC.

16.2.2 BIC

16.2.2.1 n

BIC has the same issue as AIC, in the definition

$$\text{BIC} = \text{deviance} + \log(n) \cdot p$$

p has to be the dimension of the LCM to make the mathematical argument behind BIC correct (with the same caveat that there has to be enough data left in the LCM for estimates in the LCM to be approximately normally distributed, except when the LCM is completely degenerate and has no data).

BIC also has the other issue about n, which is discussed in the help page for R function `glmbb` which we quote

It is unclear what the sample size, the n in the BIC penalty $n \log(p)$ should be. Before version 0.4 of this package the BIC was taken to be the result of applying R generic function `BIC` to the fitted object produced by R function `glm`. This is generally wrong whenever we think we are doing categorical data analysis (Kass and Raftery, 1995; Raftery, 1986). Whether we consider the sampling scheme to be Poisson, multinomial, or product multinomial (and binomial is a special case of product multinomial) the sample size is the total number of individuals classified and is the only thing that is considered as going to infinity in the usual asymptotics for categorical data analysis. Thus the option `BIC.option = "sum"` should always be used for categorical data analysis.

In more detail, if you are doing purely categorical data analysis (categorical response, all predictors categorical too, so the data can be reformatted as a contingency table) then you are not imagining the number of cells of the contingency table `length(y)` to be going to infinity, you are imagining the number of individuals classified `sum(y)` to be going to infinity. Thus `sum(y)` is the only choice for n in the BIC formula that makes sense. So the cited literature says.

Once we start thinking along these lines, it is unclear that `length(y)` which is what R function `glm` and friends do (with no option) ever makes sense. BIC was originally defined (Schwarz, 1978) for independent and identically distributed (IID) data. For that, it is clear what n means: it is the number of IID random variables in the data. For categorical data analysis, that is the number of individuals classified (the individuals are IID if they are a random sample).

BIC can be extended to non-IID data. Cavanaugh and Neath (1999) give all the gory details. But they introduce n in a completely artificial way in their assumptions. They imagine a sequence of statistical models with log likelihoods l_n all having the same parameter θ. Then they assume

$$\frac{1}{n} l_n(\theta)$$

converges (for all θ, this is their assumption (5)).

So how does that make sense for regression if we are assuming the dimension of the response vector is going to infinity, so the dimension of the predictor vectors must be going to infinity to match?
That convergence assumption implies a very restrictive condition on model matrices of the models in the sequence of models. It is not clear that this story ever makes sense. This is the reason why your humble author wrote a paper (Geyer, 2013) that does asymptotics without n. But if you take the n out of asymptotics, then BIC actually makes no sense because there is no n to go into the formula.

In summary, it is very unclear what n should be in the BIC formula when we do not have IID data. In particular, it is not clear that what R function `glm` and friends do (in particular, R generic function `BIC` does when applied to R objects of class `"glm"`, using `length(y)` rather than `sum(y)` as n) ever makes sense.

16.2.2.2 p

The case is even worse for p. The derivation of BIC (Neath and Cavanaugh, 2012) can use many different priors, but one possibility is flat within-model priors (as an approximation to Bayes factors, it does not depend on between model priors). And we know from our notes on priors for exponential families that the posterior (and hence the Bayes factor) does not exist when the MLE does not exist (in the conventional sense) and a flat prior is used.

Thus Bayes factors make no sense when one model being compared has solutions at infinity (the subject of this document). Or we might say that any model with solutions at infinity is *infinitely bad* when *flat priors are used*. But such priors are *improper* and make no philosophical sense. So this is not a problem with the models but rather with the whole idea of BIC.

16.3 AIC for Alligators

So back to alligators. We now think R is calculating AIC wrong for those models that have LCM. R does not use the same definitions of AIC and BIC that we do. It adds a constant to our definitions, that is,

$$\text{AIC} = \text{deviance} + \text{constant} + 2p$$

and similarly for BIC. The constant doesn't matter. It drops out of comparisons and calculation of weights. But we will use the same constant so we don't have to change all of the AIC values we already have.

So we recalculate AIC for these models. Because we used different formulas for R function `llmdr` we need to change these AIC values by hand (eventually this should be automated).

The whole issue is that we have different degrees of freedom output by R functions `glm` and `llmdr` and the latter are correct (so we say). Here are those degrees of freedom.

```
df.glm <- unlist(eapply(foo, function(x) x$rank))
df.llmdr <- sapply(lout, function(x) x$df)
identical(names(df.glm), names(df.llmdr))
```

```
## [1] TRUE
```

```
aic.glm <- unlist(eapply(foo, AIC))
aic.llmdr <- aic.glm + 2 * (df.llmdr - df.glm)
```

So now we have a problem that what models are within cutoff (10, the default) of the minimum are different for the two definitions, so we need a different summary for each, but we only have a function to do one.

```
min(aic.glm)
```

```
## [1] 287.9638
```

```
min(aic.llmdr)
```

```
## [1] 275.4261
```

We see that since the new minimum AIC is more than the cutoff below the old minimum AIC, that none of the models that were supposedly best under the old criterion are even on our new list.

Because we did not use R function `llmdr` as the model fitting function for R function `glmbb` we do not know that we didn't miss some models that have low AIC under the correct definition. So we don't bother to make a new table. That will have to wait until we fix R function `glmbb`.

It should come as a shock to fans of the old AIC that what is considered the best model changes, so it would change the model selection done by AIC.

Of course, model selection is a mug's game, but, even if we do model averaging, the weights change, and not just a little bit but a whole lot.

16.4 Other Ways to Specify Multinomial Data

R function `multinom` in R package `nnet` specifies multinomial response data in a different way. It has the response vector actually be a matrix or a factor. Then it doesn't need a `strata` argument. R function `llmdr` allows this too. We will illustrate the response being a factor in the next section. Here we show the response as a matrix.

```
head(alligator$food)
```

```
## [1] Fish         Invertebrate Reptile      Bird         Other
## [6] Fish
## Levels: Fish Invertebrate Reptile Bird Other
```

It looks like in these data food varies fastest, so we can reshape it into a matrix as follows,

```
food <- alligator$count
food <- matrix(food, byrow = TRUE, ncol = nlevels(alligator$food))
colnames(food) <- levels(alligator$food)
head(food)
```

```
##      Fish Invertebrate Reptile Bird Other
## [1,]    7            1       0    0     5
## [2,]    4            0       0    1     2
## [3,]   16            3       2    2     3
## [4,]    3            0       1    2     3
## [5,]    2            2       0    0     1
## [6,]   13            7       6    0     0
```

But we have to check that this is actually right, that the other variables agree for each level of this new response matrix.

```
alligator.predictors <- alligator
alligator.predictors$count <- NULL
alligator.predictors$food <- NULL
names(alligator.predictors)
```

```
## [1] "lake"   "gender" "size"
```

```
for (i in levels(alligator$food)) {
    foob <- subset(alligator.predictors, alligator$food == "Bird")
    fooi <- subset(alligator.predictors, alligator$food == i)
    rownames(foob) <- NULL
    rownames(fooi) <- NULL
    print(identical(foob, fooi))
}
```

```
## [1] TRUE
## [1] TRUE
## [1] TRUE
## [1] TRUE
## [1] TRUE
```

```
alligator.predictors <-
    subset(alligator.predictors, alligator$food == "Bird")
```

Great! We now have the data we need.

```
lout <- llmdr(food ~ lake * size, family = "multinomial",
    data = alligator.predictors)
summary(lout)
```

```
##
## Call:
## llmdr(formula = food ~ lake * size, family = "multinomial", data = alligator.predictors)
##
##                                  Estimate    GDOR Std. Error z value Pr(>|z|)
## Fish                               1.0561  0.0000     0.4105   2.573  0.01009 *
## lakeOklawaha:Fish                 -0.5452  0.0000     0.8377  -0.651  0.51516
## lakeTrafford:Fish                 -1.0561  0.0000     0.7540  -1.401  0.16132
## lakeGeorge:Fish                    0.6179  0.0000     0.7512   0.823  0.41075
## size>2.3:Fish                     -0.7196  0.0000     0.7151  -1.006  0.31427
```

```
## lakeOklawaha:size>2.3:Fish           2.2629  1.0000     1.5016   1.507  0.13183
## lakeTrafford:size>2.3:Fish           1.1896  0.0000     1.1119   1.070  0.28468
## lakeGeorge:size>2.3:Fish             0.7802  0.0000     1.1399   0.684  0.49368
## Invertebrate                        -0.6931  0.0000     0.6124  -1.132  0.25767
## lakeOklawaha:Invertebrate            1.9924  0.0000     0.8940   2.229  0.02584 *
## lakeTrafford:Invertebrate            1.4816  0.0000     0.8160   1.816  0.06943 .
## lakeGeorge:Invertebrate              2.5390  0.0000     0.8723   2.911  0.00361 **
## size>2.3:Invertebrate               -2.9444 -1.0000     1.3112  -2.246  0.02473 *
## lakeOklawaha:size>2.3:Invertebrate   3.2138  2.0000     1.9498   1.648  0.09931 .
## lakeTrafford:size>2.3:Invertebrate   2.4925  1.0000     1.5340   1.625  0.10420
## lakeGeorge:size>2.3:Invertebrate         NA  1.0000         NA      NA       NA
## Reptile                             -1.3863  0.0000     0.7906  -1.754  0.07951 .
## lakeOklawaha:Reptile                 0.2877  0.0000     1.3994   0.206  0.83712
## lakeTrafford:Reptile                 0.4700  0.0000     1.1511   0.408  0.68304
## lakeGeorge:Reptile                   0.2877  0.0000     1.3994   0.206  0.83712
## size>2.3:Reptile                    -0.2231  0.0000     1.3509  -0.165  0.86880
## lakeOklawaha:size>2.3:Reptile        2.6027  1.0000     2.1115   1.233  0.21771
## lakeTrafford:size>2.3:Reptile        1.3218  0.0000     1.7005   0.777  0.43699
## lakeGeorge:size>2.3:Reptile              NA -1.0000         NA      NA       NA
## Bird                                -1.3863  0.0000     0.7906  -1.754  0.07951 .
## lakeOklawaha:Bird                        NA -1.0000         NA      NA       NA
## lakeTrafford:Bird                   -0.2231  0.0000     1.3509  -0.165  0.86880
## lakeGeorge:Bird                      0.9808  0.0000     1.2076   0.812  0.41667
## size>2.3:Bird                        0.8755  0.0000     1.0763   0.813  0.41597
## lakeOklawaha:size>2.3:Bird               NA  2.0000         NA      NA       NA
## lakeTrafford:size>2.3:Bird           0.2231  0.0000     1.7005   0.131  0.89560
## lakeGeorge:size>2.3:Bird            -1.5686  0.0000     1.8235  -0.860  0.38966
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Maximum likelihood distribution is the limit of distributions
## with parameter vectors Estimate + s * GDOR as s goes to infinity
## (NA in the Estimate column should be read as zero).
```

When you get the data in the right format, the analysis is a lot simpler, but getting the data in the right format can be a bit of a struggle.

16.5 A Simple Multinomial Model with Completely Degenerate LCM

```
x <- seq(10, 90, 10)
y <- rep(c("red", "green", "blue"), each = 3)
y <- factor(y, levels = c("red", "green", "blue"))
data.frame(x, y)
```

```
##    x     y
## 1 10   red
## 2 20   red
## 3 30   red
## 4 40 green
## 5 50 green
## 6 60 green
## 7 70  blue
## 8 80  blue
## 9 90  blue
```

We fit these data as follows.

```
lout <- llmdr(y ~ x, family = "multinomial")
summary(lout)
```

```
##
## Call:
## llmdr(formula = y ~ x, family = "multinomial")
##
##          Estimate    GDOR Std. Error z value Pr(>|z|)
## yred           NA 10.2500         NA      NA       NA
## x:yred         NA -0.2250         NA      NA       NA
## ygreen         NA  6.5833         NA      NA       NA
## x:ygreen       NA -0.1083         NA      NA       NA
##
## Maximum likelihood distribution is the limit of distributions
## with parameter vectors Estimate + s * GDOR as s goes to infinity
## (NA in the Estimate column should be read as zero).
```

As usual, it is hard to make sense of canonical parameters, but from the fact that the `Estimates` column is all `NA` we see the LCM is completely degenerate. As usual, mean value parameters make more sense.

```
cbind(x, estimates(lout, type = "mu"))
```

```
##        x red green blue
##  [1,] 10   1     0    0
##  [2,] 20   1     0    0
##  [3,] 30   1     0    0
##  [4,] 40   0     1    0
##  [5,] 50   0     1    0
##  [6,] 60   0     1    0
##  [7,] 70   0     0    1
##  [8,] 80   0     0    1
##  [9,] 90   0     0    1
```

The low values of `x` are all red, the middle values all green, and the high values all blue. So the model can perfectly predict these data.

16.6 A Simple Multinomial Model with Partially Degenerate LCM

We change the data a little bit.

```
y[6] <- "blue"
y[7] <- "green"
y <- factor(y, levels = c("red", "green", "blue"))
data.frame(x, y)
```

```
##    x     y
## 1 10   red
## 2 20   red
## 3 30   red
## 4 40 green
## 5 50 green
## 6 60  blue
## 7 70 green
## 8 80  blue
## 9 90  blue
```

And refit

```
lout <- llmdr(y ~ x, family = "multinomial")
summary(lout)
```

```
##
## Call:
## llmdr(formula = y ~ x, family = "multinomial")
##
##             Estimate       GDOR Std. Error z value Pr(>|z|)
## yred              NA  3.800e+00         NA      NA       NA
## x:yred            NA -1.200e-01         NA      NA       NA
## ygreen    7.891e+00 -4.163e-17  6.038e+00   1.307    0.191
## x:ygreen -1.214e-01  4.626e-19  9.126e-02  -1.330    0.183
##
## Maximum likelihood distribution is the limit of distributions
## with parameter vectors Estimate + s * GDOR as s goes to infinity
## (NA in the Estimate column should be read as zero).
```

```
cbind(x, estimates(lout, type = "mu"))
```

```
##        x red      green       blue
##  [1,] 10   1 0.00000000 0.00000000
##  [2,] 20   1 0.00000000 0.00000000
##  [3,] 30   1 0.00000000 0.00000000
##  [4,] 40   0 0.95413352 0.04586648
##  [5,] 50   0 0.86069104 0.13930896
##  [6,] 60   0 0.64725931 0.35274069
##  [7,] 70   0 0.35274069 0.64725931
##  [8,] 80   0 0.13930896 0.86069104
##  [9,] 90   0 0.04586648 0.95413352
```

Now we see that the LCM is only partially degenerate because the `Estimate` column is not all `NA` but the actual numbers in the `Estimate` and `GDOR` columns don't make much sense.

But from the mean values it is clear what is going on. Now `x` fails to perfectly predict green and blue but still perfectly predicts red. The probability of green decreases with `x` and the probability of blue increases.

Of course, it looks like with such a small amount of data that the parameters of the LCM are not statistically significant, either of them. But to make a conclusion about dropping two parameters we should actually fit both models and compare.

But there doesn't seem to be any way to do that with the data in this form. It wants to use the same model matrix for each stratum, so we cannot keep `red` and `x:red` as predictors while dropping the rest. We can do that if we put the data into data frame form and use explicit strata.

```
xx <- rep(x, times = 3)
color <- rep(levels(y), each = length(x))
yy <- double(length(xx))
for (i in 1:length(x))
    yy[xx == x[i] & color == y[i]] <- 1
data.frame(x = xx, color, y = yy)
```

```
##     x color y
## 1  10   red 1
## 2  20   red 1
## 3  30   red 1
## 4  40   red 0
```

```
## 5  50   red 0
## 6  60   red 0
## 7  70   red 0
## 8  80   red 0
## 9  90   red 0
## 10 10 green 0
## 11 20 green 0
## 12 30 green 0
## 13 40 green 1
## 14 50 green 1
## 15 60 green 0
## 16 70 green 1
## 17 80 green 0
## 18 90 green 0
## 19 10  blue 0
## 20 20  blue 0
## 21 30  blue 0
## 22 40  blue 0
## 23 50  blue 0
## 24 60  blue 1
## 25 70  blue 0
## 26 80  blue 1
## 27 90  blue 1
```

Now the models we want to compare are

```
lout.alternative <- llmdr(yy ~ xx * color, family = "multinomial",
    strata = as.factor(xx))
summary(lout.alternative)
```

```
##
## Call:
## llmdr(formula = yy ~ xx * color, family = "multinomial", strata = as.factor(xx))
##
##                 Estimate       GDOR Std. Error z value Pr(>|z|)
## colorgreen     7.891e+00 -4.163e-17  6.038e+00   1.307    0.191
## colorred              NA  3.800e+00         NA      NA       NA
## xx:colorgreen -1.214e-01  4.626e-19  9.126e-02  -1.330    0.183
## xx:colorred           NA -1.200e-01         NA      NA       NA
##
## Maximum likelihood distribution is the limit of distributions
## with parameter vectors Estimate + s * GDOR as s goes to infinity
## (NA in the Estimate column should be read as zero).
```

```
data.frame(x = xx, color, y = yy, mu = estimates(lout.alternative, "mu"))
```

```
##     x color y         mu
## 1  10   red 1 1.00000000
## 2  20   red 1 1.00000000
## 3  30   red 1 1.00000000
## 4  40   red 0 0.00000000
## 5  50   red 0 0.00000000
## 6  60   red 0 0.00000000
## 7  70   red 0 0.00000000
## 8  80   red 0 0.00000000
## 9  90   red 0 0.00000000
```

```
## 10 10 green 0 0.00000000
## 11 20 green 0 0.00000000
## 12 30 green 0 0.00000000
## 13 40 green 1 0.95413352
## 14 50 green 1 0.86069104
## 15 60 green 0 0.64725931
## 16 70 green 1 0.35274069
## 17 80 green 0 0.13930896
## 18 90 green 0 0.04586648
## 19 10  blue 0 0.00000000
## 20 20  blue 0 0.00000000
## 21 30  blue 0 0.00000000
## 22 40  blue 0 0.04586648
## 23 50  blue 0 0.13930896
## 24 60  blue 1 0.35274069
## 25 70  blue 0 0.64725931
## 26 80  blue 1 0.86069104
## 27 90  blue 1 0.95413352
```

Same mean values, so same model, just expressed differently.

Now for the null model, which does have red, but not the other colors.

```
red <- as.numeric(color == "red")
lout.null <- llmdr(yy ~ xx * red, family = "multinomial",
    strata = as.factor(xx))
summary(lout.null)
```

```
##
## Call:
## llmdr(formula = yy ~ xx * red, family = "multinomial", strata = as.factor(xx))
##
##        Estimate  GDOR Std. Error z value Pr(>|z|)
## red          NA  3.80         NA      NA       NA
## xx:red       NA -0.12         NA      NA       NA
##
## Maximum likelihood distribution is the limit of distributions
## with parameter vectors Estimate + s * GDOR as s goes to infinity
## (NA in the Estimate column should be read as zero).
```

```
data.frame(x = xx, color, y = yy, mu = estimates(lout.null, "mu"))
```

```
##     x color y  mu
## 1  10   red 1 1.0
## 2  20   red 1 1.0
## 3  30   red 1 1.0
## 4  40   red 0 0.0
## 5  50   red 0 0.0
## 6  60   red 0 0.0
## 7  70   red 0 0.0
## 8  80   red 0 0.0
## 9  90   red 0 0.0
## 10 10 green 0 0.0
## 11 20 green 0 0.0
## 12 30 green 0 0.0
## 13 40 green 1 0.5
```

```
## 14 50 green 1 0.5
## 15 60 green 0 0.5
## 16 70 green 1 0.5
## 17 80 green 0 0.5
## 18 90 green 0 0.5
## 19 10  blue 0 0.0
## 20 20  blue 0 0.0
## 21 30  blue 0 0.0
## 22 40  blue 0 0.5
## 23 50  blue 0 0.5
## 24 60  blue 1 0.5
## 25 70  blue 0 0.5
## 26 80  blue 1 0.5
## 27 90  blue 1 0.5
```

Right. Predicts red perfectly, says nothing about blue or green, where "says nothing" means zero on the logit scale, or 1/2 on the probability scale, and this really does not say anything about blue or green, they are 50-50 given $x \geq 40$.

And then we have the really, really null hypothesis

```
lout.null.null <- llmdr(yy ~ red, family = "multinomial",
    strata = as.factor(xx))
summary(lout.null.null)
```

```
##
## Call:
## llmdr(formula = yy ~ red, family = "multinomial", strata = as.factor(xx))
##
##       Estimate Std. Error z value Pr(>|z|)
## red -1.570e-16  7.071e-01       0        1
```

```
data.frame(x = xx, color, y = yy, mu = estimates(lout.null.null, "mu"))
```

```
##     x color y        mu
## 1  10   red 1 0.3333333
## 2  20   red 1 0.3333333
## 3  30   red 1 0.3333333
## 4  40   red 0 0.3333333
## 5  50   red 0 0.3333333
## 6  60   red 0 0.3333333
## 7  70   red 0 0.3333333
## 8  80   red 0 0.3333333
## 9  90   red 0 0.3333333
## 10 10 green 0 0.3333333
## 11 20 green 0 0.3333333
## 12 30 green 0 0.3333333
## 13 40 green 1 0.3333333
## 14 50 green 1 0.3333333
## 15 60 green 0 0.3333333
## 16 70 green 1 0.3333333
## 17 80 green 0 0.3333333
## 18 90 green 0 0.3333333
## 19 10  blue 0 0.3333333
## 20 20  blue 0 0.3333333
## 21 30  blue 0 0.3333333
```

```
## 22 40  blue 0 0.3333333
## 23 50  blue 0 0.3333333
## 24 60  blue 1 0.3333333
## 25 70  blue 0 0.3333333
## 26 80  blue 1 0.3333333
## 27 90  blue 1 0.3333333
```

So finally the test.

```
anova(lout.null.null, lout.null, lout.alternative)
```

```
## Analysis of Deviance Table
##
## Model 1: yy ~ red
## Model 2: yy ~ xx * red
## Model 3: yy ~ xx * color
##   Df as Null Df as Alt Deviance Df for test     LRT  P-value
## 1          1         0  19.7750
## 2          0         2   8.3178           1 11.4573 0.000712
## 3          2         2   4.9560           2  3.3618 0.186207
```

These data really don't say anything about blue and green other than red versus non-red is predicted perfectly. And we cannot drop x entirely.

16.7 On the Baseline-Category Logit Parameterization

R function `multinom` in R package `nnet` uses the baseline-category logit parameterization

```
mout <- multinom(y ~ x)
```

```
## # weights:  9 (4 variable)
## initial  value 9.887511
## iter  10 value 3.153012
## iter  20 value 2.873144
## iter  30 value 2.770911
## iter  40 value 2.528968
## iter  50 value 2.522779
## iter  60 value 2.514706
## iter  70 value 2.513843
## iter  80 value 2.513066
## iter  90 value 2.512268
## iter 100 value 2.511493
## final  value 2.511493
## stopped after 100 iterations
```

```
summary(mout)
```

```
## Call:
## multinom(formula = y ~ x)
##
## Coefficients:
##       (Intercept)         x
## green   -28.42054 0.8148895
## blue    -36.33078 0.9367271
##
## Std. Errors:
##       (Intercept)        x
```

```
## green    38.71981 1.115024
## blue     39.21355 1.119270
##
## Residual Deviance: 5.022985
## AIC: 13.02299
```

```
round(predict(mout, type = "probs"), 7)
```

```
##         red     green      blue
## 1 1.0000000 0.0000000 0.0000000
## 2 0.9999945 0.0000054 0.0000000
## 3 0.9812891 0.0184491 0.0002618
## 4 0.0144584 0.9404105 0.0451312
## 5 0.0000038 0.8603669 0.1396293
## 6 0.0000000 0.6456545 0.3543455
## 7 0.0000000 0.3501492 0.6498508
## 8 0.0000000 0.1374351 0.8625649
## 9 0.0000000 0.0449963 0.9550037
```

```
estimates(lout, type = "mu")
```

```
##       red      green       blue
##  [1,]   1 0.00000000 0.00000000
##  [2,]   1 0.00000000 0.00000000
##  [3,]   1 0.00000000 0.00000000
##  [4,]   0 0.95413352 0.04586648
##  [5,]   0 0.86069104 0.13930896
##  [6,]   0 0.64725931 0.35274069
##  [7,]   0 0.35274069 0.64725931
##  [8,]   0 0.13930896 0.86069104
##  [9,]   0 0.04586648 0.95413352
```

Not too different. The differences are that R function `multinom` does not understand generic directions of recession, so can't do the right thing, and isn't even trying to do maximum likelihood, but rather what some neural net thinks is somewhat optimal. If you make allowances for extreme sloppiness, then it has the right answer, at least for the probabilities.

But what we wanted to talk about is what Agresti (2013), first page of Chapter 8, calls the baseline-category logit parameterization of the multinomial. In this example, it makes the canonical parameters

$$\log \frac{\pi_{\text{color}}(x)}{\pi_{\text{red}}(x)}$$

which makes the canonical parameters for anything red zero, hence they are dropped, which is why `summary(mout)` above does not mention red.

This is what the exponential families notes call the try II parameterization.

But R function `llmdr` does not use this parameterization. Instead it uses the symmetric parameterization, what the exponential families notes call the try III parameterization. Unlike baseline-category logit, this parameterization does not constrain any canonical parameters to be zero. Thus it is not identifiable. It is the parameterization you get when you start with Poisson sampling, and condition to get product multinomial sampling *without changing the canonical parameters*.

Why? The baseline-category logit parameterization seems easy enough to understand, and is identifiable. There is the issue of arbitrariness of which category is chosen to be "baseline", but we ignore that.

But in hindsight, considered in light of directions of recession, that parameterization seems to be a really dumb idea.

- It amounts to constraining equal to zero parameters that need to go to infinity to maximize the likelihood.
- Moreover, even if we move directions of recession to the baseline-category logit parameterization, that parameterization need not make the LCM identifiable.

We see the latter in our toy problem. R function `nnet` chooses red to be "baseline" for arbitrary reasons. But in the LCM we see that red has been conditioned out of the LCM. The LCM only has data for blue or green and $x \geq 40$. Now trying to say that red is "baseline" seems almost maximally stupid.

So OK. We cannot choose a baseline category until the LCM has already been determined. But we can then, right? No.

It is possible to construct models having LCM in which no choice of multinomial response category to be "baseline" will make the LCM identifiable.

So why this "baseline" category stuff anyway? R functions `lm` and `glm` and `llmdr` (which calls `glm.fit` to fit LCM) have no trouble figuring out which coefficients to "drop" to get an identifiable model. They don't need to be given an identifiable parameterization. So why fuss about a particular one that will only lead to insoluble problems in some instances?

Other issues with the baseline-category logit parameterization are illustrated in the next section.

16.8 Mapping Canonical to Mean Value Parameters

For product multinomial data with response vector y and strata $\mathcal{A}$, this means the subvectors y_A for $A \in \mathcal{A}$ are the independent multinomial parts of y, the saturated model usual parameters are given in terms of the canonical parameters by

$$\pi_i = \frac{e^{\theta_i}}{\sum_{j \in A} e^{\theta_j}}, \qquad i \in A \in \mathcal{A}$$

(and there is always exactly one A satisfying the condition of this equation because $\mathcal{A}$ partitions the index set of y). Then the mean value parameters are

$$\mu_i = n_A \pi_i, \qquad i \in A \in \mathcal{A}.$$

So to keep track of what is going on with the various parameterizations, we have to keep this non-identifiability of β and θ in mind.

```
theta <- estimates(lout, "theta")
theta
```

```
##        red      green blue
##  [1,]    0       -Inf -Inf
##  [2,]    0       -Inf -Inf
##  [3,]    0       -Inf -Inf
##  [4,] -Inf  3.0350690    0
##  [5,] -Inf  1.8210414    0
##  [6,] -Inf  0.6070138    0
##  [7,] -Inf -0.6070138    0
##  [8,] -Inf -1.8210414    0
##  [9,] -Inf -3.0350690    0
```

If we really, really wanted red to be the baseline category, then we could do that. Then we would have zero for the whole red column. And the blue column would be `Inf` for $x \geq 40$. And then the green column would have to be `Inf + 3.035069`, `Inf + 1.8210414`, and so forth for $x \geq 40$. What does that mean? It

means these parameters go off to infinity at the same rate, in lockstep, keeping a constant difference,

$$\begin{aligned}
\theta_{40\ \text{green}} &= \theta_{40\ \text{blue}} + 3.035069 \\
\theta_{50\ \text{green}} &= \theta_{50\ \text{blue}} + 1.8210414 \\
\theta_{60\ \text{green}} &= \theta_{60\ \text{blue}} + 0.6070138 \\
\theta_{70\ \text{green}} &= \theta_{70\ \text{blue}} - 0.6070138
\end{aligned}$$

and so forth. See what I mean when I say that the baseline-category logit parameterization causes far more pain than enlightenment?

But if we keep to the symmetric parameterization that R function `llmdr` is using, then we have to know how to use it. We go from θ to π in two steps.

```
foo <- exp(theta)
foo
```

```
##       red       green blue
##  [1,]   1  0.00000000    0
##  [2,]   1  0.00000000    0
##  [3,]   1  0.00000000    0
##  [4,]   0 20.80241248    1
##  [5,]   0  6.17828904    1
##  [6,]   0  1.83494369    1
##  [7,]   0  0.54497585    1
##  [8,]   0  0.16185711    1
##  [9,]   0  0.04807135    1
```

Then we have to make the probabilities in each stratum sum to one. Here the strata are the rows of the matrix.

```
bar <- rowSums(foo)
mu <- sweep(foo, 1, bar, "/")
mu
```

```
##       red      green       blue
##  [1,]   1 0.00000000 0.00000000
##  [2,]   1 0.00000000 0.00000000
##  [3,]   1 0.00000000 0.00000000
##  [4,]   0 0.95413352 0.04586648
##  [5,]   0 0.86069104 0.13930896
##  [6,]   0 0.64725931 0.35274069
##  [7,]   0 0.35274069 0.64725931
##  [8,]   0 0.13930896 0.86069104
##  [9,]   0 0.04586648 0.95413352
```

Of course, R function `estimates` does this for us, so we don't have to understand this to get the numbers, but this is what it is doing to get them.

```
estimates(lout, "pi")
```

```
##       red      green       blue
##  [1,]   1 0.00000000 0.00000000
##  [2,]   1 0.00000000 0.00000000
##  [3,]   1 0.00000000 0.00000000
##  [4,]   0 0.95413352 0.04586648
##  [5,]   0 0.86069104 0.13930896
##  [6,]   0 0.64725931 0.35274069
##  [7,]   0 0.35274069 0.64725931
```

```
##  [8,]   0 0.13930896 0.86069104
##  [9,]   0 0.04586648 0.95413352
```

16.9 DOR, DOC, and GDOR for Multinomial

Usually `family = "multinomial"` means product multinomial, so we first need to start with the symmetric parameterization for product multinomial sampling described above.

We use a theorem described in Section 3.17 of Geyer (2009), which says, if you can find DOR or GDOR for Poisson sampling, then they also work for multinomial or product multinomial sampling (actually Geyer (2009) leaves the product multinomial case as an exercise for the reader, but it is fully worked out in the design document for R package `llmdr` (Geyer, 2022), which is found in every installation of the package).

This theorem needs one idea. The indicator vectors for strata are vectors u having coordinates

$$u_i = \begin{cases} 1, & i \in A \\ 0, & i \notin A \end{cases}$$

where $A \in \mathcal{A}$ are the sets of indices for strata, as in the section on multinomial parameterization above.

Then the exact statement of the theorem is that the model matrix for Poisson sampling has to have all the columns of the model matrix for product multinomial sampling plus all the indicator vectors for multinomial strata as columns. Then every MLE for the Poisson model is also an MLE for the product multinomial model (when both use the same parameterization, that is, the product multinomial is using the symmetric parameterization) and every DOR for the Poisson model is a DOR for the product multinomial model.

The only difference is that the parameterization for Poisson sampling is (usually) identifiable, and the parameterization for product multinomial sampling is never identifiable. Its directions of constancy are exactly the indicator vectors for strata described above.

Thus we can immediately drop all components of the Poisson parameter vector that correspond to indicator vectors of strata, and similarly for components of DOR. They are not identifiable. This leaves us with MLE for β and GDOR for product multinomial sampling.

Thus we only know how to solve multinomial problems indirectly, by converting them to Poisson problems, and then converting back at the end.

A much more complete characterization of product multinomial DOR, DOC, and GDOR is found in the design document for R package `llmdr` cited above.

Bibliography

Agresti, A. (2013) *Categorical Data Analysis*. Hoboken, NJ: John Wiley & Sons.

Allaire, J. J., Xie, Y., McPherson, J., et al. (2021) *R Package `rmarkdown`: Dynamic Documents for R, Version 2.11*. Available at: https://rmarkdown.rstudio.com.

Barndorff-Nielsen, O. E. (1978) *Information and Exponential Families*. Chichester, England: Wiley.

Brown, L. D. (1986) *Fundamentals of Statistical Exponential Families with Applications in Statistical Decision Theory*. Hayward, CA: Institute of Mathematical Statistics.

Cavanaugh, J. E. and Neath, A. A. (1999) Generalizing the derivation of the Schwarz information criterion. *Communications in Statistics — Theory and Methods*, **28**, 49–66. DOI: 10.1080/03610929908832282.

Eck, D. J. and Geyer, C. J. (2021) Computationally efficient likelihood inference in exponential families when the maximum likelihood estimator does not exist. *Electronic Journal of Statistics*, **15**, 2105–2156. DOI: 10.1214/21-EJS1815.

Gelius-Dietrich, G. (2021) *R Package* `glpkAPI`*: R Interface to c Api of Glpk, Version 1.3.3*. Available at: https://CRAN.R-project.org/package=glpkAPI.

Geyer, C. J. (1990) *Likelihood and exponential families*. PhD thesis. University of Washington. Available at: http://purl.umn.edu/56330.

Geyer, C. J. (2009) Likelihood inference in exponential families and directions of recession. *Electronic Journal of Statistics*, **3**, 259–289. DOI: 10.1214/08-EJS349.

Geyer, C. J. (2013) Asymptotics of maximum likelihood without the LLN or CLT or sample size going to infinity. In *Advances in Modern Statistical Theory and Applications: A Festschrift in Honor of Morris L. Eaton* (eds G. L. Jones and X. Shen), pp. 1–24. Hayward, CA: Institute of Mathematical Statistics. DOI: 10.1214/12-IMSCOLL1001.

Geyer, C. J. (2020) *R Package* `glmbb`*: All Hierarchical or Graphical Models for Generalized Linear Model, Version 0.5-1*. Available at: http://cran.r-project.org/package=glmbb.

Geyer, C. J. (2022) *R Package* `llmdr`*: Log-Linear Models Done Right, Version 0.1*. Available at: http://github.com/cjgeyer/llmdr/.

Geyer, C. J. and Thompson, E. A. (1992) Constrained Monte Carlo maximum likelihood for dependent data (with discussion). *Journal of the Royal Statistical Society, Series B*, **54**, 657–699. DOI: 10.1111/j.2517-6161.1992.tb01443.x.

Geyer, C. J., Meeden, G. D. and Fukuda, K. (2021) *R Package* `rcdd`*: Computational Geometry, Version 1.4*. Available at: http://cran.r-project.org/package=rcdd.

Gilbert, P. and Varadhan, R. (2019) *R Package* `numDeriv`*: Accurate Numerical Derivatives, Version 2016.8-1.1*. Available at: https://CRAN.R-project.org/package=numDeriv.

Kass, R. E. and Raftery, A. E. (1995) Bayes factors. *Journal of the American Statistical Association*, **90**, 773–795. DOI: 10.1080/01621459.1995.10476572.

Lucas, A., Scholz, I., Boehme, R., et al. (2021) *R Package* `gmp`*: Multiple Precision Arithmetic, Version 0.6-2.1*. Available at: https://CRAN.R-project.org/package=gmp.

Maechler, M. (2021) *R Package* `sfsmisc`*: Utilities from 'Seminar Fuer Statistik' ETH Zurich, Version 1.1-12*. Available at: https://CRAN.R-project.org/package=sfsmisc.

Neath, A. A. and Cavanaugh, J. E. (2012) The Bayesian information criterion: Background, derivation, and applications. *WIREs Computational Statistics*, **4**, 199–203.

Raftery, A. E. (1986) A note on bayes factors for log-linear contingency table models with vague prior information. *Journal of the Royal Statistical Society, Series B*, **48**, 249–250. DOI: 10.1111/j.2517-6161.1986.tb01408.x.

Schwarz, G. (1978) Estimating the dimension of a model. *Annals of Statistics*, **6**, 461–464. DOI: 10.1214/aos/1176344136.

Varadhan, R. (2015) *R Package* `alabama`*: Constrained Nonlinear Optimization, Version 2015.3-1*. Available at: https://CRAN.R-project.org/package=alabama.

Venables, W. N. and Ripley, B. D. (2002) *Modern Applied Statistics with S*. Fourth. New York: Springer. Available at: https://www.stats.ox.ac.uk/pub/MASS4/.

Venables, W. N. and Ripley, B. D. (2021a) *R Package* `MASS`*: Support Functions and Datasets for Venables and Ripley's MASS, Version 7.3-54*. Available at: https://CRAN.R-project.org/package=MASS.

Venables, W. N. and Ripley, B. D. (2021b) *R Package* `nnet`*: Feed-Forward Neural Networks and Multinomial Log-Linear Models, Version 7.3-16*. Available at: https://CRAN.R-project.org/package=nnet.

Xie, Y., Allaire, J. J. and Grolemund, G. (2018) *R Markdown: The Definitive Guide*. Chapman; Hall/CRC. Available at: https://bookdown.org/yihui/rmarkdown.

Xie, Y., Dervieux, C. and Riederer, E. (2020) *R Markdown Cookbook*. Chapman; Hall/CRC. Available at: https://bookdown.org/yihui/rmarkdown-cookbook.