1 Confidence Intervals for Mean Value Parameters

For complete separation example of Agresti (2013, Section 6.5.1), we need confidence intervals. The theory in Geyer (2009) says a $100(1-\alpha)\%$ confidence region for the parameter is the set of all parameter values that put at least probability α on the observed data vector. This is valid only for the "complete separation" case. See Geyer (2009) and Eck and Geyer (submitted) for when "complete separation" does not hold.

The probability in question is

$$\prod_{i=1}^{n} p_i^{y_i} (1 - p_i)^{1 - y_i}$$

but the computer does not like 0^0 so we change this to

$$\left(\prod_{\substack{i \in N \\ y_i = 1}} p_i \right) \left(\prod_{\substack{i \in N \\ y_i = 0}} 1 - p_i \right)$$

where N is the index set for y and θ, $N = \{1, \ldots, n\}$. To avoid underflow, we take logs.

$$\left(\sum_{\substack{i \in N \\ y_i = 1}} \log(p_i) \right) + \left(\sum_{\substack{i \in N \\ y_i = 0}} \log(1 - p_i) \right) \tag{1}$$

We want to consider this a function of the submodel parameter β (called "coefficients" by R). The relation is

$$\theta = M\beta$$

where M is the model matrix, and

$$p = \operatorname{logit}^{-1}(\theta)$$

where this inverse logit function operates componentwise

$$\begin{aligned} p_i &= \frac{e^{\theta_i}}{1 + e^{\theta_i}} = \frac{1}{1 + e^{-\theta_i}} \\ 1 - p_i &= \frac{1}{1 + e^{\theta_i}} = \frac{e^{-\theta_i}}{1 + e^{-\theta_i}} \end{aligned}$$

for all i. Taking logs gives

$$\begin{aligned} \log(p_i) &= \theta_i - \log(1 + e^{\theta_i}) = -\log(1 + e^{-\theta_i}) \\ \log(1 - p_i) &= -\log(1 + e^{\theta_i}) \quad = -\theta_i - \log(1 + e^{-\theta_i}) \end{aligned}$$

We want to maximize and minimize components of p over the region where (1) is at least $\log(\alpha)$. Actually, it may be more computationally stable to maximize and minimize components of θ over the same region. Since p is a componentwise monotone function of θ, both amount to the same thing.

For any quantity Q we have the chain rule

$$\frac{\partial Q}{\partial \beta_j} = \sum_{i=1}^{n} \frac{\partial Q}{\partial \theta_i} \frac{\partial \theta_i}{\partial \beta_j} = \sum_{i=1}^{n} \frac{\partial Q}{\partial \theta_i} m_{ij}$$

where m_{ij} are the components of the model matrix M. So it suffices to worry about derivatives with respect to the θ's.

First

$$\frac{\partial p_i}{\partial \theta_i} = p_i(1 - p_i)$$

and $\partial p_i / \partial \theta_j = 0$ for $i \neq j$.

Let $g(\beta)$ denote the value of (1). Then in case $y_i = 1$ we have

$$\frac{\partial g(\beta)}{\partial \theta_i} = \frac{\partial \log(p_i)}{\partial \theta_i} = 1 - p_i$$

and in case $y_i = 0$ we have

$$\frac{\partial g(\beta)}{\partial \theta_i} = \frac{\partial \log(1 - p_i)}{\partial \theta_i} = -p_i$$

Make the data.

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

Try to fit the data.

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

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

summary(gout)

##
## Call:
## glm(formula = y ~ x, family = binomial, x = TRUE)
##
## Deviance Residuals:
##        Min          1Q      Median          3Q         Max
## -1.045e-05  -2.110e-08   0.000e+00   2.110e-08   1.045e-05
```

```
##
## 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
```

Code up these functions.

```
modmat <- gout$x

f <- function(beta, k, ...) {
    stopifnot(is.numeric(beta))
    stopifnot(is.finite(beta))
    stopifnot(length(beta) == ncol(modmat))
    stopifnot(is.numeric(k))
    stopifnot(is.finite(k))
    stopifnot(length(k) == 1)
    stopifnot(as.integer(k) == k)
    stopifnot(k %in% 1:nrow(modmat))
    theta <- modmat %*% beta
    ifelse(y == 1, theta, - theta)[k]
}

df <- function(beta, k, ...) {
    stopifnot(is.numeric(beta))
    stopifnot(is.finite(beta))
    stopifnot(length(beta) == ncol(modmat))
    stopifnot(is.numeric(k))
    stopifnot(is.finite(k))
    stopifnot(length(k) == 1)
    stopifnot(as.integer(k) == k)
    stopifnot(k %in% 1:nrow(modmat))
    ifelse(y == 1, 1, -1)[k] * as.vector(modmat[k, ])
}
```

OK. We are ready to test f and df. Let us make up some points at which to test it.

```
n <- length(x)
p <- length(gout$coefficients)

beta.test <- rep(0, p)
library(numDeriv)
df(beta.test, 1)

## [1]  -1 -10

grad(f, beta.test, k = 1)

## [1]  -1 -10

for (i in 1:n)
    print(all.equal(df(beta.test, i), grad(f, beta.test, k = i)))

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

for (j in 1:5) {
    beta.test <- rnorm(p) * 0.2
    for (i in 1:n)
        print(all.equal(df(beta.test, i), grad(f, beta.test, k = i)))
}

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

Seems to be OK. Now for g and dg.

```
g <- function(beta, alpha, ...) {
    stopifnot(is.numeric(beta))
    stopifnot(is.finite(beta))
    stopifnot(length(beta) == ncol(modmat))
    stopifnot(is.numeric(alpha))
    stopifnot(length(alpha) == 1)
    stopifnot(0 < alpha && alpha < 1)
    eta <- modmat %*% beta
    logp <- ifelse(eta < 0, eta - log1p(exp(eta)), - log1p(exp(- eta)))
    logq <- ifelse(eta < 0, - log1p(exp(eta)), - eta - log1p(exp(- eta)))
    logpboundary <- ifelse(y == 1, logp, logq)
    sum(logpboundary) - log(alpha)
}
dg <- function(beta, alpha, ...) {
    stopifnot(is.numeric(beta))
    stopifnot(is.finite(beta))
    stopifnot(length(beta) == ncol(modmat))
    stopifnot(is.numeric(alpha))
```

```r
    stopifnot(length(alpha) == 1)
    stopifnot(0 < alpha && alpha < 1)
    theta <- modmat %*% beta
    pp <- ifelse(theta < 0, exp(theta) / (1 + exp(theta)),
        1 / (1 + exp(- theta)))
    qq <- ifelse(theta < 0, 1 / (1 + exp(theta)),
        exp(- theta) / (1 + exp(- theta)))
    # apparently R function auglag wants the jacobian of
    # the inequality constraints to be a matrix
    # in this case since g returns a vector of length 1
    # this function should return a 1 by p matrix
    result <- ifelse(y == 1, qq, - pp) %*% modmat
    dimnames(result) <- NULL
    result
}
```

```r
alpha.test <- 0.05
beta.test <- rep(0, p)
library(numDeriv)
dg(beta.test, alpha.test)

##      [,1] [,2]
## [1,]    0  100

grad(g, beta.test, alpha = alpha.test)

## [1]   0 100

all.equal(dg(beta.test, alpha.test),
    rbind(grad(g, beta.test, alpha = alpha.test)))

## [1] TRUE

for (j in 1:5) {
    beta.test <- rnorm(p) * 0.2
    print(all.equal(dg(beta.test, alpha.test),
        rbind(grad(g, beta.test, alpha = alpha.test))))
}

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

Everything looks good. Let's find confidence limits.

```
library(alabama)
beta.start <- c(0, 0)
bounds <- rep(NA_real_, length(y))
for (i in seq(along = bounds)) {
    aout <- auglag(beta.start, f, df, g, dg,
        control.outer = list(trace = FALSE),
        k = i, alpha = 0.05)
    if (aout$convergence == 0)
        bounds[i] <- aout$value
}
bounds

## [1]  0.9185667  0.4303787 -0.2852351 -2.9440131 -2.9440330 -0.2852351
## [7]  0.4303787  0.9185668
```

Fix up `bounds` so they correspond to what we are interested in.

```
bounds <- ifelse(y == 1, bounds, - bounds)
bounds

## [1] -0.9185667 -0.4303787  0.2852351  2.9440131 -2.9440330 -0.2852351
## [7]  0.4303787  0.9185668

bounds.lower.theta <- ifelse(y == 1, bounds, -Inf)
bounds.upper.theta <- ifelse(y == 1, Inf, bounds)
data.frame(x, y, lower = bounds.lower.theta, upper = bounds.upper.theta)

##    x y      lower      upper
## 1 10 0       -Inf -0.9185667
## 2 20 0       -Inf -0.4303787
## 3 30 0       -Inf  0.2852351
## 4 40 0       -Inf  2.9440131
## 5 60 1 -2.9440330        Inf
## 6 70 1 -0.2852351        Inf
## 7 80 1  0.4303787        Inf
## 8 90 1  0.9185668        Inf

bounds.lower.p <- 1 / (1 + exp(- bounds.lower.theta))
bounds.upper.p <- 1 / (1 + exp(- bounds.upper.theta))
data.frame(x, y, lower = bounds.lower.p, upper = bounds.upper.p)

##    x y      lower     upper
## 1 10 0 0.00000000 0.2852500
## 2 20 0 0.00000000 0.3940359
## 3 30 0 0.00000000 0.5708292
## 4 40 0 0.00000000 0.9499798
## 5 60 1 0.05001929 1.0000000
## 6 70 1 0.42917079 1.0000000
## 7 80 1 0.60596409 1.0000000
## 8 90 1 0.71474999 1.0000000
```

Now make the plot.

```r
par(mar = c(4, 4, 0, 0) + 0.1)
plot(x, y, axes = FALSE, type = "n",
    xlab = expression(x), ylab = expression(mu(x)))
segments(x, bounds.lower.p, x, bounds.upper.p, lwd = 2)
box()
axis(side = 1)
axis(side = 2)
points(x, y, pch = 21, bg = "white")
```

Our Figure 1 is Figure 2 in Eck and Geyer (submitted). It is also the second figure in the talk (first is scatterplot of data).

2 Support of Submodel Canonical Statistic

The course notes Geyer (2016) show how to visualize the support of the submodel canonical statistic $M^T y$ where M is the model matrix of an exponential family GLM and y is the response vector. We dump that in here to get that figure. There is a lot of unnecessary blather here that is copied verbatim from those notes. We apologize for that but don't want to edit it for this. We only want the figure (Figure 4 below) for the talk.

We will use the theory of Barndorff-Nielsen completions of exponential families from Geyer (2009).

2.1 Support Points

That theory tells us that we must look at the set of all possible values of the canonical statistic $M^T y$ where M is the model matrix and y is the response vector. For the model, M has two columns: the first column is all ones (the "intercept" column) and the second column is x. So let's find that set. There are 2^n possible values where n is the dimension of the response vector because each component of y can be either zero or one. The following code makes all of those vectors.

```r
yy <- NULL
n <- length(y)
for (i in 1:n) {
    j <- 2^(i - 1)
    k <- 2^n / j / 2
    yy <- cbind(rep(rep(0:1, each = j), times = k), yy)
}
```

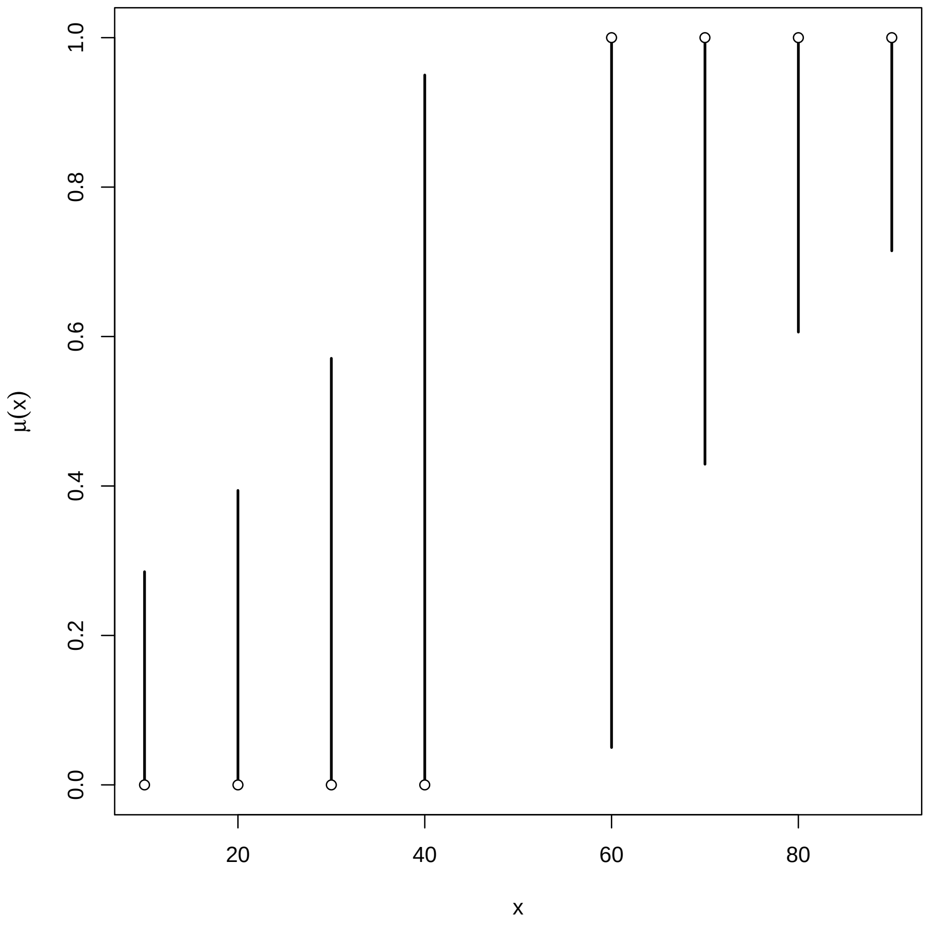

Figure 1: One-sided 95% confidence intervals for mean value parameters. Bars are the intervals. Vertical axis is the probability of observing response value one when the predictor value is x. Solid dots are the observed data.

```
head(yy)

##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]    0    0    0    0    0    0    0    0
## [2,]    0    0    0    0    0    0    0    1
## [3,]    0    0    0    0    0    0    1    0
## [4,]    0    0    0    0    0    0    1    1
## [5,]    0    0    0    0    0    1    0    0
## [6,]    0    0    0    0    0    1    0    1

dim(yy)

## [1] 256   8
```

For those who know how to count in binary, row i is $i-1$ expressed in binary. Have you heard the joke: there are two kinds of people in this world, those who divide everything into two kinds and those who don't? And its nerd version: there are 10 kinds of people in this world, those who know binary and those who don't?

For those who don't, the following code shows that every row of `yy` is different, every row contains only zeros and ones, and there are 2^n rows.

```
fred <- apply(yy, 1, paste, collapse = "")
head(fred)

## [1] "00000000" "00000001" "00000010" "00000011" "00000100" "00000101"

length(unique(fred)) == length(fred)

## [1] TRUE

all(apply(yy, 1, function(x) all(x %in% 0:1)))

## [1] TRUE

nrow(yy) == 2^n

## [1] TRUE
```

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)
```

Figure 2 shows these possible values of the submodel canonical statistic.

And now we are stuck. Figure 2 seems to show that the observed data vector is an extreme value, but we cannot easily figure out the direction of recession.

2.2 Tangent Vectors

Vectors $Y(\omega) - y$, where y is the observed value of the canonical statistic vector and $Y(\omega)$ are other possible values of the canonical statistic vector, are called *tangent vectors* (Geyer, 2009, explains the reason they have this name). If

$$V = \{ v_i : i \in I \}$$

is the set of tangent vectors, then the set of a nonnegative combinations of them, vectors of the form

$$\sum_{i \in A} a_i v_i$$

where A is a finite set and $a_i \geq 0$ for all i, is called the *tangent cone*. It is denoted $\text{con}(\text{pos}\, T)$ in Geyer (2009).

Load library `rcdd`

```
library(rcdd)

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

Figure 3 shows the tangent vectors and tangent cone. The points in Figures 2 and 3 are the same except in Figure 3 they are moved so the one corresponding to the observed value of the canonical statistic is the origin $(0, 0)$. The gray area is the tangent cone (set of all nonnegative combinations of tangent vectors).

We are interested in the case where a finite subset of the tangent vectors gives the same tangent cone, that is, when S is a finite subset of T such that $\text{con}(\text{pos}\, S) = \text{con}(\text{pos}\, T)$. This is obviously the case, when the statistical model has finite support so T is finite, as in logistic regression and log-linear models for contingency tables with multinomial or product multinomial sampling. As we shall see, it is also the case for Poisson regression with log link and for log-linear models for contingency tables with Poisson sampling.

For generalized linear models (GLM) we do not need all the tangent vectors. For the saturated model, tangent vectors $Y(\omega) - y$ such that $Y(\omega)$ and y differ only in one coordinate are enough to generate the whole tangent cone (Geyer, 2009, Section 3.11). Moreover, if V_{sat} is a set of vectors generating the tangent cone for the saturated model, then

$$V_{\text{sub}} = \{ M^T v : v \in V_{\text{sat}} \}$$

is a set of vectors generating the tangent cone for the canonical affine submodel with model matrix M (Geyer, 2009, Section 3.10).

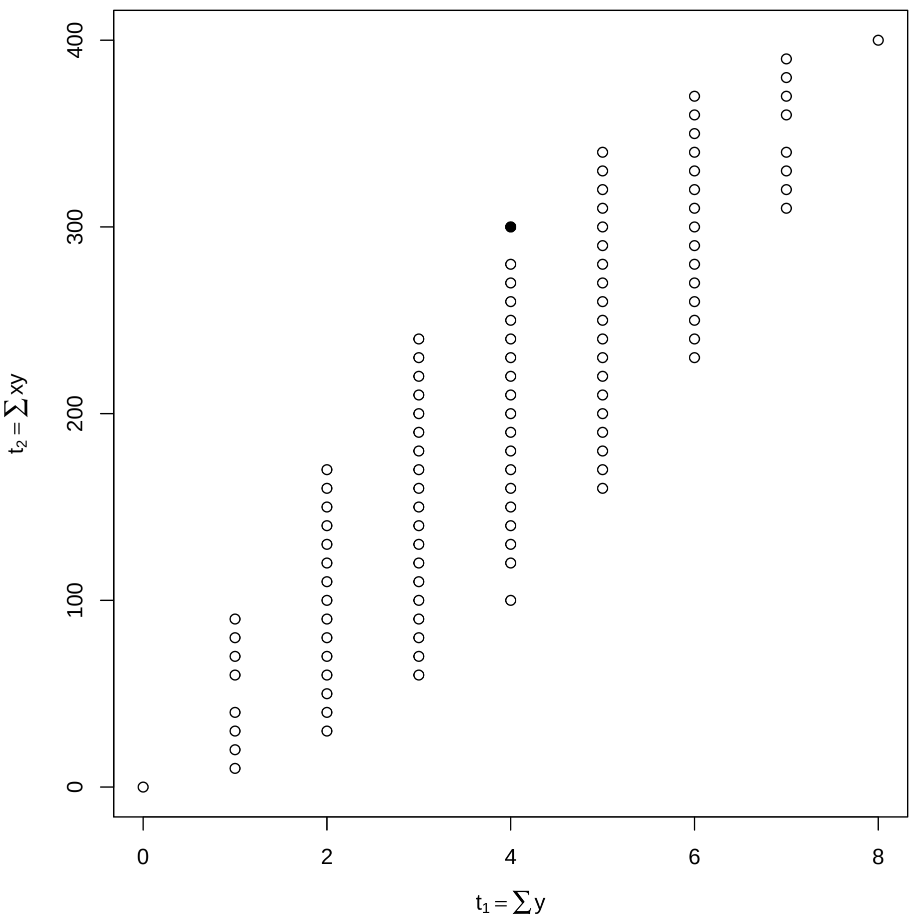

Figure 2: Possible values of the submodel canonical statistic vector $M^T y$ for the data shown in Figure 1. Solid dot is the observed value of the submodel canonical statistic vector.

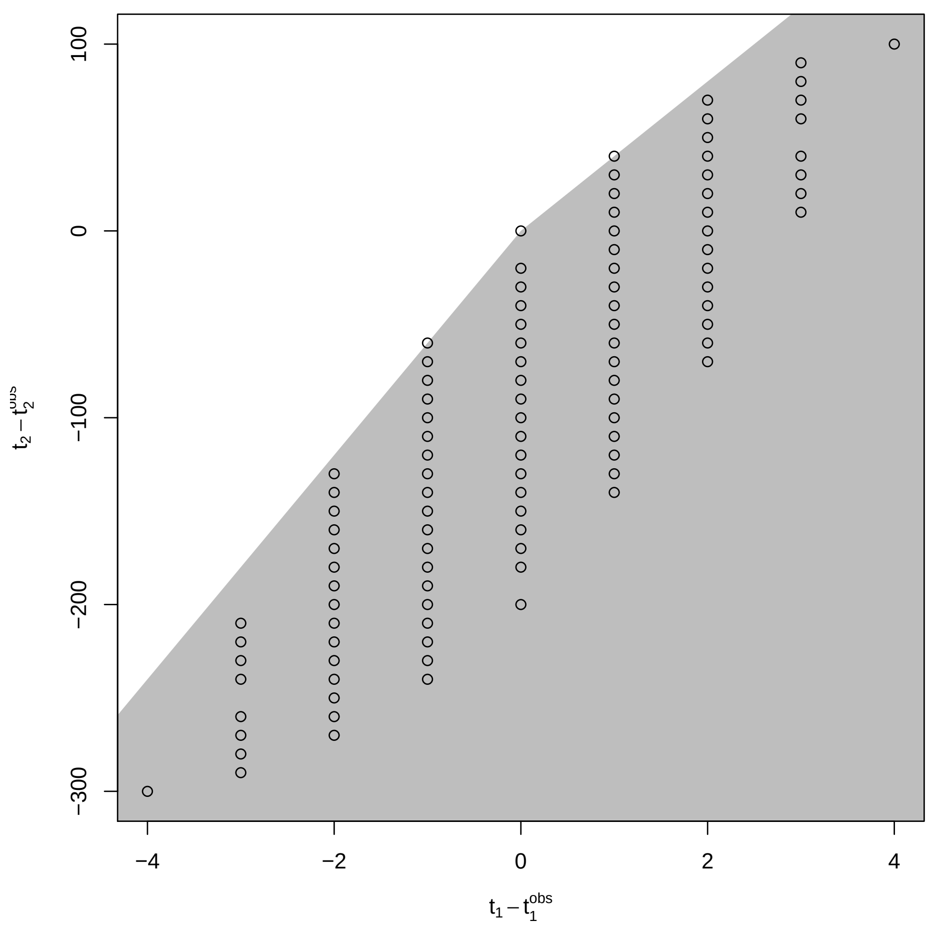

Figure 3: Tangent vectors and tangent cone for data shown in Figure 1. Dots are tangent vectors, gray region is tangent cone.

So now we need to learn how to find these tangent vectors for the saturated model.

First consider logistic regression when y has Bernoulli components (zero-or-one-valued). If the observed value of y_i is 0, the only other possible value is 1, so the vector e_i, which has all coordinates equal to 0 except the i-th component, which is 1, is a tangent vector. Similar reasoning says $-e_i$ is a tangent vector if $y_i = 1$.

Second consider logistic regression when y has binomial components. Now we have not only components y_i of the response vector but sample sizes n_i that go with them. We see that if $y_i = 0$, then e_i is a tangent vector (as before), and if $y_i = n_i$, then $-e_i$ is a tangent vector (as before), but now we also have the case $0 < y_i < n_i$ in which case it is possible to change the i-th coordinate either up or down, so both e_i and $-e_i$ are tangent vectors.

Third consider Poisson regression with log link. Just like in the binomial case we have e_i is a tangent vector when $y_i = 0$, and both e_i and $-e_i$ are tangent vectors when $0 < y_i < \infty$. Since there is no upper bound to the range of a Poisson random variable, there is no case where only $-e_i$ is a tangent vector.

2.3 Calculating the Linearity

We now want to calculate a GDOR, but that calculation proceeds in two steps in the algorithm of Geyer (2009). First we need to find the *linearity* of the tangent cone (Geyer, 2009, Section 3.12), which is the smallest vector subspace contained in the tangent cone, although Geyer (2009) also (somewhat sloppily) uses the same term for a set of vectors spanning this vector subspace.

If V_{sub} is a set of vectors generating the tangent cone for the canonical affine submodel, then there is an R function `linearity` in the R package `rcdd` that calculates

$$L_{\text{sub}} = \{\, v \in V_{\text{sub}} : -v \in \operatorname{con}(\operatorname{pos} V_{\text{sub}})\,\}. \tag{2}$$

This is the set of all the given tangent vectors that are in the linearity of the tangent cone. They also span it, hence determine it.

If we use L_{sub} as defined in (2) to denote a set of tangent vectors. Then the linearity considered as a vector space is denoted $\operatorname{span} L_{\text{sub}}$.

The linearity is useful for three reasons. The hyperplane H defined that supports the limiting conditional model (LCM), which is defined in Theorem 6 in Geyer (2009) and the discussion following it, can be expressed as $H = y + \operatorname{span} L_{\text{sub}}$. So the linearity tells us the support of the LCM. We also need to know what the linearity is in order to calculate a GDOR. Finally, the linearity tells whether the MLE exists or not. It exists if and only if $L_{\text{sub}} \neq V_{\text{sub}}$ (Geyer, 2009, Theorem 4).

So let us calculate the linearity for our example, the data shown in Figure 1. We follow Section 4.1 of Geyer (2008).

```
tanv <- m
tanv[y == 1, ] <- (-tanv[y == 1, ])
```

```
vrep <- makeV(rays = tanv)
lout <- linearity(d2q(vrep), rep = "V")
lout

## integer(0)
```

$M^T e_i$ is just the i-th row of M, so the rows of `m` are either tangent vectors or -1 times tangent vectors. So we assign `tanv` to be `m` and then adjust the signs. For rows of `m` such that corresponding component of `y` is equal to one, we need to change the sign. So the second and third lines of the code chunk above make `tanv` a matrix whose rows are the elements of $V_{\texttt{sub}}$. Then next two lines are idiomatic usage of the R package `rcdd`. The result `lout` is an integer vector giving the indices of the tangent vectors in the linearity, that is, `tanv[linearity, ]` is a basis for the linearity considered as a vector subspace.

Here the result is a vector of length zero, which says the empty set of vectors spans the linearity, which means it is the trivial vector subspace $\{0\}$ that has only one point. We could actually see this in Figure 3, the gray area is a pointed cone, so it contains only the trivial subspace.

So this tells us that the support of the LCM for this example contains only one point. The MLE distribution is completely degenerate, concentrated at y. The MLE distribution says the only data we could have observed is what we did observe; no other data values were possible. Before anyone decides this is weird, let me remind you this is only an estimate, and, as always, estimates are not parameters. This degeneracy causes no problem so long as we don't overinterpret it.

This complete degeneracy of the MLE distribution is what Agresti calls "complete separation."

2.4 Calculating Generic Directions of Recession

If $L_{\text{sub}} \neq V_{\text{sub}}$ the MLE does not exist in the original model (OM), and we need to calculate a GDOR. In this we follow Geyer (2009, Section 3.13). A vector η in the parameter space is a GDOR if and only if

$$\langle v, \eta \rangle = 0, \qquad v \in L_{\text{sub}} \tag{3a}$$
$$\langle v, \eta \rangle < 0, \qquad v \in V_{\text{sub}} \setminus L_{\text{sub}} \tag{3b}$$

and we can find one such η by solving the following linear program

$$\begin{aligned} &\text{maximize} \\ &\qquad \varepsilon \\ &\text{subject to} \\ &\qquad\qquad \varepsilon \le 1 \\ &\qquad \langle v, \eta \rangle = 0, \qquad\qquad v \in L_{\text{sub}} \\ &\qquad \langle v, \eta \rangle \le -\varepsilon, \qquad\quad v \in V_{\text{sub}} \setminus L_{\text{sub}} \end{aligned} \tag{4}$$

where η is a vector, ε is a scalar, and (η, ε) denotes a vector of length one more than the length of η. This vector is the vector of variables of the linear program. The η part of the solution is a generic direction of recession. The ε part does not matter.

So we solve this linear program to calculate the GDOR, still following Section 4.1 of Geyer (2008).

```
p <- ncol(tanv)
hrep <- cbind(0, 0, -tanv, -1)
hrep <- rbind(hrep, c(0, 1, rep(0, p), -1))
objv <- c(rep(0, p), 1)
pout <- lpcdd(d2q(hrep), d2q(objv), minimize = FALSE)
names(pout)

## [1] "solution.type"   "primal.solution" "dual.solution"   "optimal.value"

pout$solution.type

## [1] "Optimal"

gdor <- q2d(pout$primal.solution[1:p])
gdor

## [1] -5.0  0.1
```

The code chunk above is not general. It assumes the linearity is trivial, as in the particular example we are working on. More on this later.

So now we have a GDOR, we should put that on the plot, but we cannot. The reason is that η is a vector in the parameter space (as we have been saying over and over), but the space plotted in Figure 2 is the sample space for the canonical statistic vector. (I tried. There is no way to draw η into Figure 2.) What we can do is add the hyperplane

$$H = \{ \, x \in \mathbb{R}^2 : \langle x, \eta \rangle = \langle y, \eta \rangle \, \} \tag{5}$$

See Figure 4. The fact that the only possible value of the canonical statistic vector that is on H is the observed value y again tells us that the LCM is completely degenerate, concentrated at the observed value.

References

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

Eck, D. J., and Geyer, C. J. Computationally efficient likelihood inference in exponential families when the maximum likelihood estimator does not exist. Submitted to *Annals of Statistics*. https://arxiv.org/abs/1803.11240

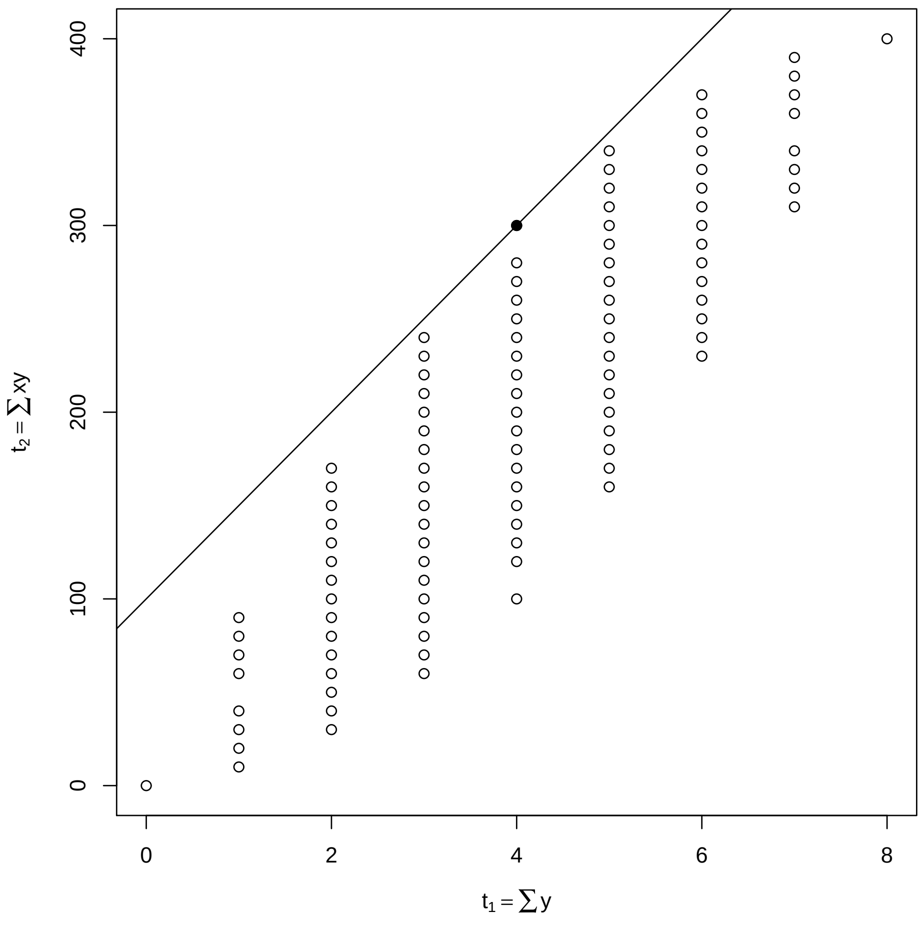

Figure 4: Possible values of the submodel canonical statistic vector $M^T y$ for the data shown in Figure 1. Solid dot is the observed value of the submodel canonical statistic vector. Solid line is the hyperplane (5) on which the LCM is concentrated.

Geyer, C. J. (2008). Supporting theory and data analysis for "Likelihood inference in exponential families and directions of recession". Technical Report 672, School of Statistics, University of Minnesota. `http:www.stat.umn.edu/geyer/gdor/phaseTR.pdf`.

Geyer, C. J. (2009). Likelihood inference in exponential families and directions of recession. *Electronic Journal of Statistics*, **3**, 259–289.

Geyer, C. J. (2016). Two Examples of Agresti. Class notes, PDF `http://www.stat.umn.edu/geyer/8931expfam/infinity.pdf`, knitr source `http://www.stat.umn.edu/geyer/8931expfam/infinity.Rnw`

Geyer, C. J. (2018). Fast Valid Statistical Inference when the Maximum Likelihood Estimate Does Not Exist in an Exponential Family Model and the "Usual Asymptotics" are Bogus. Talk at Mini-Conference to Celebrate Elizabeth Thompson's Contributions to Statistics, Genetics and the University of Washington, June 19, 2018. `http://users.stat.umn.edu/~geyer/ElizabethFest/`