Design of an R Package to do Log-Linear Models (including Logistic Regression, Poisson Regression with Log Link, and Multinomial Response Models) Providing Valid Statistical Inference when Maximum Likelihood Estimates Do Not Exist

Charles J. Geyer

February 3, 2024

Contents

1 License

2 R

- The version of R used to make this document is 4.3.2.
- The version of the `knitr` package used to make this document is 1.45.
- The version of the `Matrix` package used to make this document is 1.6.5.
- The version of the `glpkAPI` package used to make this document is 1.3.4.
- The version of the `gmp` package used to make this document is 0.7.4.
- The version of the `rcdd` package used to make this document is 1.6.

- The version of the `CatDataAnalysis` package used to make this document is 0.1.5.
- The version of the `alabama` package used to make this document is 2023.1.0.
- The version of the `numDeriv` package used to make this document is 2016.8.1.1.
- The version of the `sfsmisc` package used to make this document is 1.1.17.
- The version of the `mcmc` package used to make this document is 0.9.8.

```r
library("Matrix")
library("glpkAPI")

## using GLPK version 4.65

library("rcdd")

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

library("CatDataAnalysis")
library("alabama")

## Loading required package:  numDeriv

library("sfsmisc")
library("mcmc")
set.seed(42)
```

We do not load R package `gmp` because it overrides functions in R packages `Matrix` and `base` that we may need. We are especially worried about it redefining R functions `%*%` and `apply`.

3 Introduction

This is the design document for an R package to fit

- log-linear models for categorical data (Agresti, 2013, Chapter 9), which include, as special cases

- logistic regression (logit link) (Agresti, 2013, Chapter 5),
- Poisson regression with log link (Agresti, 2013, Section 4.3), and
- multinomial response models in which the response is nominal-valued (not ordered categorical, exponential family canonical link) (Agresti, 2013, Section 8.1).

We aim to deal with these models just as well as existing R software, such as R functions `glm` and `loglin` in the R core, R function `loglm` in R package `MASS`, and R function `multinom` in R package `nnet` (these two packages being R recommended packages that are installed by default in every R installation).

But, even better, we aim to do the Right Thing when the MLE does not exist in the given model but rather in the Barndorff-Nielsen completion of the model (Geyer, 2009). And doing the Right Thing means having R generic functions that work with regression results, like `summary`, `anova`, `predict`, and `confint`, also do the Right Thing.

Existing R software does an abysmal job when the MLE does not exist. R function `glm` does check for such a situation, but uses checks that are weak and give false positives and false negatives. R function `loglin` uses the mean value parameterization only and hence provides (up to inaccuracy of computer arithmetic) valid maximum likelihood estimates of mean value parameters even when maximum likelihood estimates of canonical parameters do not exist, provided the user correctly gives a starting value with the correct pattern of structural zeros (if any).

No R packages known to us implement valid methods of statistical inference in such situations.

The technical report that goes with Geyer (2009) does show how to do such valid statistical inference using R package `rcdd`, which is on CRAN (Geyer, et al., 2023). But the code is very slow on big data, and it is not user friendly. Eck and Geyer (2021) describe faster methods. Here we describe even faster and more reliable methods.

We call this package `llmdr` for "log-linear models done right" which is a rip-off of the title of the book *Linear Algebra Done Right* (Axler, 2024).

Unlike R function `glm` we only fit statistical models that are regular full exponential families. There is no choice of link functions.

4 Exponential Family Theory

A statistical model is an *exponential family of distributions* if it has a log likelihood of the form

$$l(\theta) = \langle y, \theta \rangle - c(\theta) \tag{1}$$

where

- y is a vector statistic, called the *canonical statistic*,
- θ is a vector parameter, called the *canonical parameter*,
- c is called the *cumulant function*, and
- $\langle \cdot, \cdot \rangle$ is a bilinear form placing two vector spaces in duality

(Geyer, 2009). The last means

$$\langle y, \theta \rangle = \textstyle\sum_i y_i \theta_i$$

when we consider vectors to be tuples (like R does).

A *canonical affine submodel* of an exponential family has parameterization of the form

$$\theta = a + M\beta \tag{2}$$

where

- a is a known vector called the *offset vector*,
- M is a known matrix called the *model matrix*, and
- β is the submodel parameter vector

(Geyer, Wagenius, and Shaw, 2007). Barndorff-Nielsen (1978, Section 8.2) used the term "affine hypothesis" for the same concept. Here "known" means not considered random. Either a or M may depend on covariates, but all statistical inference is done conditional on covariates, as is usual for regression-like statistical models. M is also called the *design matrix*, but R does not use this terminology, so we don't either. R function `glm` calls β the *coefficients* vector and calls y the *response* vector.

When we need to contrast a canonical affine submodel with the model having log likelihood (1) that it is a submodel of, we call the latter the *saturated model*, because it is the largest possible canonical affine submodel (the one with model matrix equal to the identity matrix).

Generalized linear models (GLM) that are exponential families (logistic regression and Poisson regression with log link) are a special case. Geyer, et al. (2007) call them affine models rather than linear models because (2) makes θ an affine function of β.

A canonical affine submodel is itself an exponential family with log likelihood

$$\begin{aligned} l(\beta) &= \langle y, M\beta \rangle - c(a + M\beta) \\ &= \langle M^T y, \beta \rangle - c_{\text{sub}}(\beta) \end{aligned}$$

and hence

- submodel canonical statistic $M^T y$,
- submodel canonical parameter β, and
- submodel cumulant function c_{sub}

(Geyer, et al., 2007, Section 2.4; Geyer, 2009, Section 3.9).

Theoretically, there would be no problem in not considering vectors as tuples, but rather as elements of an abstract vector space as defined in linear algebra and then replacing (2) with an arbitrary known affine function between vector spaces, although then there is a little difficulty in defining what $M^T y$ means (M is the the derivative of the affine function that replaces (2), that derivative being a linear function as defined in functional analysis (Lang, 1993, Chapter XIII, Section 2), and M^T is its adjoint linear function as defined in functional analysis (Lang, 1993, Chapter V, Section 2)).

R package `aster` (Geyer, 2023) actually does this. It makes y a matrix and M a three-way array. But this turned out to be a design mistake that limited the generality of models the package could handle and made dealing with model matrices a pain for users. R package `aster2` (Geyer, 2017) did not repeat this design mistake. In fact, R package `aster` has had many changes to ameliorate this design mistake (accepting either matrices or three-way arrays as "model matrices").

Not only users dislike this sort of thing, but R also dislikes it. R wants y and β to be what it considers vectors, and it wants M to be what it considers a matrix, the result returned by R function `model.matrix` or `sparse.model.matrix`. The dimension of y is the row dimension of M, and the dimension of β is the column dimension of M. The offset vector a, when present, has the the same dimension as the response vector y.

The R functions discussed above also do this. Collectively, they allow y to be a vector or matrix or factor or array or table and allow β to be a vector or matrix or nothing (R functions `loglin` and `loglm` fit canonical affine submodels using the iterative proportional fitting (IPF) algorithm, which makes no explicit use of any of the mathematical objects in (2)). Our experience with R package `aster` tells us this is also a design mistake, but (as with R package aster) it is a design mistake that is impossible to change for reasons of backward compatibility.

Since we are starting with a clean slate, we need to be careful to do something sane.

5 Another Look at Models

As is well known, the term log-linear model does not specify a statistical model. Different sampling schemes are possible. These are

- Poisson,
- multinomial, and
- product multinomial.

In all three the components of the response vector are counts. There is another sampling scheme

- binomial,

but this is a special case of product multinomial (although our code will do this as a special case).

In Poisson sampling the components of the response vector are assumed to be independent Poisson random variables (having different means, which are determined by the offset vector and model matrix, which in turn is determined by the model formula and covariates).

In multinomial sampling the response vector is assumed to have a multinomial distribution. R function `loglin` in the R core does not use formulas to specify these models, but R function `loglm` in R package `MASS` does use formulas to specify them.

In product multinomial sampling, the response vector is assumed to be partitioned into subvectors that have independent multinomial distributions. R function `multinom` in R package `nnet` uses formulas to specify these models. It specifies the partition by making the response vector a matrix: the rows of the matrix are independent multinomial random vectors. We will allow arbitrary partitions.

The elements of the partition we call *strata*, as in stratified random sampling.

These three sampling schemes are not always distinguished because likelihood ratio tests and Pearson chi-squared tests of goodness of fit give the same test statistics and P-values for any of these sampling schemes. This is taught even in intro statistics classes. When doing categorical data analysis of a two-dimensional table, intro stats teaches two hypothesis tests

- the test of independence (of the random variable whose values are the row labels of the table and the random variable whose values are the column labels of the table) and

- the test of homogeneity of proportions where each row (resp. column) of the table is assumed to be an independent multinomial random vector, that is, the row (resp. column) totals were fixed by experimental design (not random). The null hypothesis is that each of these multinomial distributions has the same parameter vector.

The first is the multinomial sampling scheme, and the second is the product multinomial sampling scheme, although intro stats books do not call them that (nor do they mention Poisson sampling). Ditto for R function `chisq.test` which does these tests when given a matrix of counts as its first argument and does not distinguish between Poisson, multinomial, or product multinomial sampling.

It is a general theorem of categorical data analysis that these three sampling schemes lead to identical point estimates of the mean value parameter vector (the vector of expectations of components of the response vector, the vector of mean cell counts). They also lead to identical test statistics for likelihood ratio tests and Pearson chi-squared tests of goodness of fit. Thus the sampling scheme can often be ignored.

But not always ignored. The different sampling schemes have models of different dimensions, so point estimates of mean value parameters are the same but sampling distributions of those point estimates are different. Also, Rao and Wald test statistics are different. So we will have to be careful to distinguish the sampling schemes when necessary.

6 Existing Model Specification

Existing R software dealing with these models uses quite complicated model specification.

The simplest model specification is for those models this package shares with R function `glm`: logistic regression and Poisson regression with log link. The formula determines the model matrix, and in all cases the response vector is or can be converted to a vector of counts. When the response is a factor (binomial family), the response vector is zero-or-one-valued, zero indicating the first level of the factor (this is a bizarre convention but the one used by R function `glm`). When the response is a matrix (binomial family), it must have two columns, and the response vector is the first column, and the row sums are the binomial sample sizes.

After that model specifications get complicated.

- For R function `multinom` in R package `nnet` the response "vector" cannot be a vector; it must be a matrix or a factor.

- For R function `loglin` the response "vector" must be an array (or an object of class `"table"` but such an object is also an array). There is no formula. Instead there is a specification of which margins of the array have observed values matched to maximum likelihood expected values in estimation of the model. We have to decide whether we want any similar non-formula specification for our package.

- For R function `loglm` in R package `MASS` the response "vector" can be an array as for R function `loglin` but can also be a vector, in either case a formula is used in conjunction with the "shape" of the response "vector" but the meaning of the formula is radically different for the two shapes. In R-4.1.0 R function `loglm` calls R function `loglin` to do its model fitting, but, of course, this could be changed in future versions.

All of these models we consider to have product multinomial sampling, but we will call this `family = "multinomial"` following the precedent of R function `glm` calling product binomial sampling `family = "binomial"`.

7 Parameterization of Multinomial Models

We use the symmetric parameterization of multinomial models, which is not identifiable. Let y be the response vector having index set I, and let $\mathcal{A}$ be a partition of I. Then the model assumes that the subvectors y_A, $A \in \mathcal{A}$ are independent multinomial random vectors (this is the most general form of product multinomial sampling).

If θ is the canonical parameter vector of the saturated model (called the "linear predictor" in GLM parlance), then the cell probabilities are

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

and the cell expected values are

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

where n_A is the sample size for stratum A.

Agresti (2013, Section 8.1.1) uses the baseline category logit parameterization to obtain identifiability. Like R function `multinom` in R package `nnet`, he lets y be a matrix whose rows are the strata. Then he keeps (2) but has every term be a matrix, that is, θ and a and $M\beta$ are matrices having the same shape as y. This makes β a matrix, whose row dimension is the column dimension of M and whose column dimension is the size of the strata (the column dimension of y).

Then our other equations above must be matrix equations

$$\pi_{ij} = \frac{e^{\theta_{ij}}}{\sum_{k \in J} e^{\theta_{ik}}}$$

where J is the column index set of θ and β and the response matrix, and

$$\mu_{ij} = n_i \pi_{ij}.$$

The baseline category logit parameterization constrains the first column of β considered as a matrix to be the zero vector. This makes the first column of θ considered as a matrix to also be the zero vector. This is problematic.

- It is not obvious what the "first" element of each stratum is in general stratified sampling.
- Constraining certain parameters to be zero in advance of determining whether the MLE exists in the conventional sense considerably complicates that, as we shall see in Section 8 below.
- Worse. This response is a matrix stuff makes every stratum the same size, which is a serious limitation on model specification. Not every application will be covered. So this is really just a bad idea.

Thus we do not worry about identifiability yet and use the symmetric parameterization. We will eventually have to adopt some constraints to force identifiability, but do so just before model fitting.

In order to have unified notation in the rest of the document, we will stick to the general product multinomial sampling scheme. The response vector is a vector, so is the offset vector, and the submodel canonical parameter vector β. The strata are given by an arbitrary partition $\mathcal{A}$ of the index set of the response vector.

None of this β is a matrix stuff. Our software will have to deal with that, but our theory does not have to deal with that.

8 Determining Whether an MLE Exists

8.1 Directions of Recession

From Section 3.9 of Geyer (2009) we see that if y is the response vector and M is the model matrix, then $M^T y$ is the submodel canonical statistic vector. Theorems 1, 3, and 4 in Geyer (2009) say that an MLE exists for the submodel canonical parameter vector (called the coefficients vector by R function `glm`) if and only if every direction of recession of the log likelihood is also a direction of constancy, where, if Y denotes a random realization of the response vector and y the observed value of the response vector, a vector δ in the submodel canonical parameter space is

- a *direction of recession* (DOR) if and only if $(Y - y)^T M \delta \leq 0$ almost surely and
- a *direction of constancy* (DOC) if and only if $(Y - y)^T M \delta = 0$ almost surely.

The set of all DOR is denoted $N_{C_{\text{sub}}}(M^T y)$. For this notation see Geyer (2009), Section 3.2, Theorem 3, and Sections 3.9 and 3.10. A DOR δ is generic (is a GDOR) if $N_{C_{\text{sub}}}(M^T y)$ is not a vector subspace and δ is in the relative interior of $N_{C_{\text{sub}}}(M^T y)$ (Geyer, 2009, Section 3.6).

An MLE does not exist in the OM if and only if $N_C(M^T y)$ is not a vector subspace (Geyer, 2009, Theorem 4). The relative interior of a nonempty convex set is always nonempty (Geyer, 2009, Section 3.6). Hence an MLE does not exist in the OM if and only if a GDOR exists.

8.2 Poisson Sampling

Theorem 1. *If y is the observed value of the response vector and M is the model matrix for a Poisson model and $\eta = M\delta$, then δ is a DOR if and only if both of the following conditions hold,*

$$\eta_i \leq 0, \qquad \textit{for all } i,$$

and

$$\eta_i < 0 \text{ implies } y_i = 0, \qquad \textit{for all } i.$$

A direction δ is a DOC if and only if $M\delta = 0$.

Proof. From the preceding section, δ is a DOR if and only if $(Y - y)^T \eta \leq 0$ almost surely.

If both conditions of the theorem statement hold, then $(Y - y)^T \eta \leq 0$ almost surely, because Y_i is nonnegative-integer-valued.

Conversely, suppose one or the other of the conditions of the theorem statement fails to hold. If $\eta_i > 0$ for some i, then there is positive probability that $Y_i > y_i$ and $Y_j = y_j$ for $j \neq i$. But when that event occurs we have $(Y - y)^T \eta = (Y_i - y_i)\eta_i > 0$, and δ cannot be a direction of recession. If $\eta_i < 0$ and $y_i > 0$ for some i, then there is positive probability that $Y_i < y_i$ and $Y_j = y_j$ for $j \neq i$. But when that event occurs we have $(Y - y)^T \eta = (Y_i - y_i)\eta_i > 0$, and δ cannot be a direction of recession.

Suppose $\eta = M\delta \neq 0$, so there exists an i such that $\eta_i \neq 0$. It is possible that $Y_i \neq y_i$ and $Y_j = y_j$ for $j \neq i$. But when that event occurs we have $(Y - y)^T \eta = (Y_i - y_i)\eta_i \neq 0$, so δ cannot be a DOC. Conversely, if $\eta = M\delta = 0$, then $\langle Y - y, \eta \rangle = 0$ and δ is a DOC. $\square$

Theorem 2. *In the same situation as in Theorem 1, let I be the index set of the response vector, and let I^{**} be the set of $i \in I$ for which there exists a DOR δ such that $\eta = M\delta$ and $\eta_i < 0$. Then a DOR δ is a GDOR if and only if I^{**} is nonempty, $\eta = M\delta$, and $\eta_i < 0$ for all $i \in I^{**}$.*

A GDOR fails to exist and an MLE exists in the OM if and only if I^{**} is empty.

Proof. With the definition of I^{**} in the theorem statement, $N_{C_{\text{sub}}}(M^T y)$ is the set of all δ such that $\eta = M\delta$ and

$$\eta_i \leq 0, \qquad i \in I^{**} \tag{3a}$$

$$\eta_i = 0, \qquad i \in I \setminus I^{**} \tag{3b}$$

Being the solution set of a finite set of linear equalities and inequalities, $N_{C_{\text{sub}}}(M^T y)$ is a polyhedral convex cone (Geyer, 2009, Section 3.5). It follows that the relative interior of $N_{C_{\text{sub}}}(M^T y)$ is the set of all δ such that $\eta = M\delta$ and

$$\begin{aligned} \eta_i &< 0, \qquad i \in I^{**} \\ \eta_i &= 0, \qquad i \in I \setminus I^{**} \end{aligned}$$

This follows from Proposition 2.42 in Rockafellar and Wets (1998) plus the fact that for a half space the relative interior is the same as the interior plus the fact that the inequalities (3a) do not collectively imply an equality (by definition of I^{**}). □

We can search for DOR with linear programming. Consider the following linear programming problem. Let I be the index set of the response vector and the saturated model canonical parameter vector, let J be the index set of the submodel parameter vector, and let m_{ij} denote the components of the model matrix. Let $I^* = \{\, i \in I : y_i = 0 \,\}$, where y is the observed value of the response vector.

$$\begin{aligned} \text{minimize } & \sum_{i \in I^*} \sum_{j \in J} m_{ij} \delta_j \\ \text{subject to } & \sum_{j \in J} m_{ij} \delta_j = 0, \qquad i \in I \setminus I^* \\ & -1 \le \sum_{j \in J} m_{ij} \delta_j \le 0, \qquad i \in I^* \end{aligned} \tag{4}$$

Here δ is the (vector) variable in the linear program, δ_j are its components, J is its index set, and m_{ij} are the components of the model matrix M.

Theorem 3. *An MLE exists in the OM if and only if the linear program (4) has optimal value zero. When the optimal value is negative, it must be less than or equal to -1, the solution δ is a DOR that is not a DOC, and taking limits in that direction gives a limiting conditional model having smaller support than the original model.*

See Theorem 6 and the following discussion in Geyer (2009) for discussion of limits in directions of recession and the resulting limiting conditional models.

Proof. It is clear from Theorem 1 that the feasible region (the set of δ satisfying the constraints) of the linear program (4) contains some positive scalar multiple of every DOR (the reason why it does not include every DOR is to be a bounded region so the linear program has a solution) and contains no vectors that are not DOR.

Optimal solutions exist, because the feasible region is nonempty (it always contains the zero vector) and because the feasible region is bounded in the direction of the gradient of the objective function, and every solution is a DOR. If the optimal value is zero, then the solution set is the null space of the model matrix, so the solution is a DOC, and an MLE does exist in the OM (but is not unique if the solution is nonzero).

If a DOR δ exists that is not a DOC, then then we can rescale this δ so that every component of $\eta = M\delta$ is between -1 and 0 (inclusive) and some component is equal to -1. For this rescaled δ the objective function is less than or equal to -1. Hence the optimal value of the linear program must be less than or equal to -1, and the solution must also be a DOR that is not a DOC.

The support of the LCM taking limits in the direction δ that is the solution of the linear program is

$$\{\, y : \eta_i < 0 \text{ implies } y_i = 0 \,\}$$

where $\eta = M\delta$. When the optimal value is negative, this is clearly smaller than the support of the OM. □

Although this linear program is guaranteed to find a DOR that is not a DOC if one exists, it is not guaranteed to find a GDOR. To do that, we need to solve multiple linear programs. Let I^{***} be any nonempty subset of the I^* defined just before the linear program (4), and consider the linear program.

$$\begin{array}{lll} \text{minimize} & \displaystyle\sum_{i \in I^{***}} \sum_{j \in J} m_{ij} \delta_j & \\ \text{subject to} & \displaystyle\sum_{j \in J} m_{ij} \delta_j = 0, & i \in I \setminus I^* \\ & -1 \le \displaystyle\sum_{j \in J} m_{ij} \delta_j \le 0, & i \in I^{***} \\ & \displaystyle\sum_{j \in J} m_{ij} \delta_j \le 0, & i \in I^* \setminus I^{***} \end{array} \tag{5}$$

Theorem 4. *A DOR δ exists such that $\eta = M\delta$ satisfies $\eta_i < 0$ for some $i \in I^{***}$ if and only if the linear program (5) has negative optimal value,* *in which case that optimal value must be less than or equal to -1 and the solution is such a DOR.*

Proof. The feasible region of the linear program (5) still contains only DOR and contains some positive scalar multiple of every DOR. Hence the solution is a DOR.

If the optimal value is zero, then we have proved that no DOR δ has $\eta = M\delta$ and $\eta_i < 0$ for some $i \in I^{***}$.

If the optimal value is negative, then we can rescale the solution δ so that $\eta = M\delta$ has every η_i for $i \in I^{***}$ is between -1 and 0 (inclusive) and one of these η_i is equal to -1. For this rescaled δ the objective function is less than or equal to -1. Hence the optimal value of the linear program must be less than or equal to -1. □

Algorithm 1 Find GDOR, Poisson Sampling

```
Set $I^{**} = \varnothing$
Set $I^{***} = \{ i \in I : y_i = 0 \}$
Set $\gamma = 0$
**repeat** {
    Solve the linear program (5)
    **if** (linear program has no solution) **error**
    **if** (optimal value is zero) **break**
    Set $\delta$ to be the solution of the linear program
    Set $\gamma = \gamma + \delta$
    Set $\eta = M\delta$
    Set $I^{**} = I^{**} \cup \{ i \in I : \eta_i < 0 \}$
    Set $I^{***} = I^{***} \setminus I^{**}$
    **if** ($I^{***} = \varnothing$) **break**
}
```

Theorem 5. *Algorithm 1 constructs the set I^{**} of Theorem 2, and, if that set is nonempty, constructs a GDOR γ.*

If the algorithm stops after the first linear program because it had optimal value zero, then it stops with $\gamma = 0$, which is, of course, a DOR that is a DOC and not a GDOR.

Another way to describe the set I^{**} is that it is the set $\{ i \in I : \eta_i < 0 \}$ where $\eta = M\gamma$.

The set I^{**} in the program is, of course, not the set I^{**} in Theorem 2 until the algorithm terminates. One can think of it as the set of indices that we have established to be in I^{**} of the theorem at that point of the computation.

Proof. If the result is that I^{**} is empty, then from Theorem 1 the MLE exists in the OM. Otherwise, the algorithm continues searching for components of I^{*} that should be in I^{**} until all are found. Since a positive linear combination of DOR is another DOR, the result γ is a DOR that satisfies the conditions of Theorem 2 to be a GDOR. □

8.3 Binomial Sampling

The situation becomes only slightly more complicated for binomial sampling. Now we not only have the response vector y but also a sample size vector n with $0 \le y_i \le n_i$ for all i.

The following theorems are similar to those in the preceding section. Since the proofs are also similar, we omit them.

Theorem 6. *For a binomial model, if y is the observed value of the response vector, n is the sample size vector, M is the model matrix, and $\eta = M\delta$, then δ is a DOR if and only if both of the following conditions hold,*

$$\eta_i > 0 \text{ implies } y_i = n_i, \qquad \textit{for all } i.$$

and

$$\eta_i < 0 \text{ implies } y_i = 0, \qquad \textit{for all } i.$$

Theorem 7. *In the same situation as in Theorem 6, let I be the index set of the response vector, and let I^{**} be the set of $i \in I$ for which there exists a DOR δ such that $\eta = M\delta$ and $\eta_i \neq 0$. Then a DOR δ is a GDOR if and only if I^{**} is nonempty, $\eta = M\delta$, and $\eta_i \neq 0$ for all $i \in I^{**}$.*

Let $I^{*} = \{\, i \in I : y_i = 0 \,\text{or}\, y_i = n_i \,\}$, where y is the observed value of the response vector, and let I^{***} be any nonempty subset of I^{*}.

For binomial we need the following linear program.

$$\begin{aligned}
\text{minimize}\quad & \sum_{\substack{i \in I^{***} \\ y_i = 0}} \sum_{j \in J} m_{ij} \delta_j - \sum_{\substack{i \in I^{***} \\ y_i = n_i}} \sum_{j \in J} m_{ij} \delta_j \\
\text{subject to}\quad & \sum_{j \in J} m_{ij} \delta_j = 0, && i \in I \setminus I^* \\
& -1 \le \sum_{j \in J} m_{ij} \delta_j \le 0, && i \in I^{***} \text{ and } y_i = 0 \\
& 0 \le \sum_{j \in J} m_{ij} \delta_j \le 1, && i \in I^{***} \text{ and } y_i = n_i \\
& \sum_{j \in J} m_{ij} \delta_j \le 0, && i \in I^* \setminus I^{***} \text{ and } y_i = 0 \\
& 0 \le \sum_{j \in J} m_{ij} \delta_j, && i \in I^* \setminus I^{***} \text{ and } y_i = n_i
\end{aligned} \tag{6}$$

Theorem 8. *When $I^{***} = I^*$, an MLE exists in the OM if and only if the linear program (6) has optimal value zero. When the optimal value is negative, it must be less than or equal to -1, the solution δ is a DOR that is not a direction of constancy, and taking limits in that direction gives a limiting conditional model having smaller support than the original model.*

Theorem 9. *A DOR δ exists such that $\eta = M\delta$ satisfies $\eta_i \neq 0$ for some $i \in I^{***}$ if and only if the linear program (6) has negative optimal value, in which case that optimal value must be less than or equal to -1 and the solution is such a DOR.*

Theorem 10. *Algorithm 2 (page 20) constructs the set I^{**} of Theorem 7, and, if that set is nonempty, constructs a GDOR γ.*

8.4 Multinomial Sampling

Let I be the index set of a Poisson GLM having response vector Y and let $\mathcal{A}$ be a partition of I. For any $A \in A$, let

$$N_A = \sum_{i \in A} Y_i \tag{7}$$

and let N denote the random vector having components N_A, $A \in \mathcal{A}$.

Algorithm 2 Find GDOR, Binomial Sampling

```
Set $I^{**} = \varnothing$
Set $I^{***} = \{\, i \in I : y_i = 0 \text{ or } y_i = n_i \,\}$
Set $\gamma = 0$
repeat {
    Solve the linear program (6)
    if (linear program has no solution) error
    if (optimal value is zero) break
    Set $\delta$ to be the solution of the linear program
    Set $\gamma = \gamma + \delta$
    Set $\eta = M\delta$
    Set $I^{**} = I^{**} \cup \{\, i \in I : \eta_i \neq 0 \,\}$
    Set $I^{***} = I^{***} \setminus I^{**}$
    if ($I^{***} = \varnothing$) break
}
```

Theorem 11. *Consider a Poisson GLM having response vector Y and mean value parameter vector μ. The conditional distribution of Y given $N = n$, where N is defined by (7), is product multinomial with cell probabilities*

$$\pi_i = \frac{\mu_i}{\sum_{j \in A} \mu_j}, \qquad i \in A \in \mathcal{A},$$

and mean value parameter vector having components

$$\mu_i = n_A \pi_i, \qquad i \in A \in \mathcal{A}. \tag{8}$$

When $\mathcal{A} = \{I\}$ is the trivial partition, we say the sampling model is "multinomial" rather than "product multinomial".

Proof. The probability mass function (PMF) of the Poisson model is

$$f_\mu(y) = \prod_{i \in I} \frac{\mu_i^{y_i}}{y_i!} e^{-\mu_i}$$

Because sum of independent Poisson is Poisson, the marginal distribution of N is

$$f_\mu(n) = \prod_{A \in \mathcal{A}} \left(\sum_{i \in A} \mu_i \right)^{n_A} \exp\left(- \sum_{i \in A} \mu_i \right) \frac{1}{n_A!}$$

Hence the conditional PMF of Y given N is

$$f_\mu(y \mid n) = \prod_{A \in \mathcal{A}} \frac{n_A!}{\left(\sum_{i \in A} \mu_i\right)^{n_A}} \prod_{i \in A} \frac{\mu_i^{y_i}}{y_i!} = \prod_{A \in \mathcal{A}} \binom{n_A}{y_A} \prod_{i \in A} \left(\frac{\mu_i}{\sum_{j \in A} \mu_j} \right)^{y_i}$$

where we have introduced the notation y_A for the "subvector" having index set A and components y_i and the multinomial coefficient

$$\binom{n_A}{y_A} = n_A! \prod_{i \in A} \frac{1}{y_i!}$$

This conditional PMF is the product of multinomial PMF's. $\square$

Theorem 12. *For a product multinomial model induced by a partition $\mathcal{A}$ having observed value of the canonical statistic vector y and model matrix M, a vector δ is a DOR if and only if $\eta = M\delta$ and the following condition holds*

$$i \in A \in \mathcal{A} \text{ and } \eta_i < \max_{j \in A} \eta_j \text{ implies } y_i = 0 \tag{9}$$

And a vector δ is a DOC if and only if

$$i \in A \in \mathcal{A} \text{ implies } \eta_i = \max_{j \in A} \eta_j \tag{10}$$

Proof. The vector δ is a DOR if and only if $\langle Y - y, \eta \rangle \leq 0$ almost surely. Define

$$\zeta_A = \max_{i \in A} \eta_i, \qquad A \in \mathcal{A}. \tag{11}$$

Then

$$\begin{aligned} \langle Y - y, \eta \rangle &= \sum_{A \in \mathcal{A}} \sum_{i \in A} (Y_i - y_i) \eta_i \\ &= \sum_{A \in \mathcal{A}} \sum_{i \in A} (Y_i - y_i)(\eta_i - \zeta_A) \end{aligned}$$

holds almost surely because

$$\sum_{i \in A} Y_i = \sum_{i \in A} y_i = n_A, \qquad \text{almost surely.}$$

If (9) holds, since $\eta_i - \zeta_A \neq 0$ implies $\eta_i - \zeta_A < 0$ and $Y_i - y_i \geq 0$, we have $\langle Y - y, \eta \rangle \leq 0$.

Conversely, suppose (9) fails to hold, that is, there exists an $i \in A \in \mathcal{A}$ such that $\eta_i < \zeta_A$ but $y_i > 0$. Then there is positive probability that $Y_i < y_i$

and $Y_j = y_j$ for $j \neq i$. When that event occurs we have $\langle Y - y, \eta \rangle = (Y_i - y_i)\eta_i > 0$, and δ cannot be a DOR.

A vector δ is a DOC if and only if $\langle Y - y, \eta \rangle = 0$ almost surely.

If (10) holds then $i \in A$ implies $\eta_i - \zeta_A = 0$, so $\langle Y - y, \eta \rangle = 0$.

Conversely, if (10) fails to hold, then there exist $i \in A \in \mathcal{A}$ such that $\eta_i < \zeta_A$. And there is positive probability that $Y_i \neq y_i$ and $Y_j = y_j$ for $j \neq i$. When that event occurs we have $\langle Y - y, \eta \rangle = (Y_i - y_i)\eta_i \neq 0$, and δ cannot be a DOC. $\square$

Theorem 13. *For a product multinomial model induced by a partition $\mathcal{A}$ having observed value of the canonical statistic vector y and model matrix M, a vector δ is a GDOR if and only if the following condition holds.*

Define the set I^{**} as the set of $i \in I$ such that there exists a DOR δ such that $\eta = M\delta$ and

$$\eta_i < \max_{j \in A} \eta_j, \qquad \text{for the } A \in \mathcal{A} \text{ such that } i \in A.$$

Then a vector δ is a GDOR if and only if I^{**} is nonempty and $\eta = M\delta$ and

$$i \in A \in \mathcal{A} \text{ and } i \in I^{**} \text{ implies } \eta_i < \max_{j \in A} \eta_j. \tag{12}$$

Proof. From Theorem 12 we know the set of all DOR is the set of all δ such that $\eta = M\delta$ satisfies (9). The set of all DOR is a closed convex cone (Geyer, 2009, Theorem 3 and Section 3.2).

Now we rewrite (9) as

$$\bigcap_{A \in \mathcal{A}} \bigcap_{i \in I^{**} \cap A} \bigcup_{j \in A \setminus \{i\}} \{\, \eta \in \mathbb{R}^I : \eta_i \leq \eta_j \,\} \tag{13a}$$

and (12) as

$$\bigcap_{A \in \mathcal{A}} \bigcap_{i \in I^{**} \cap A} \bigcup_{j \in A \setminus \{i\}} \{\, \eta \in \mathbb{R}^I : \eta_i < \eta_j \,\} \tag{13b}$$

Clearly (13a) is a closed set and (13b) is an open set. If (13b) is nonempty, then it is the relative interior of (13a) by the relative interior criterion (Rockafellar and Wets, 1998, Exercise 2.41). So it only remains to be shown that (13b) is nonempty. We know for every $i \in I^{**}$ there exists a vector $\eta = M\delta$ such that

$$\eta_i < \max_{j \in A} \eta_j$$

where A is the unique element of $\mathcal{A}$ containing i. Let H be the set of all such η (one for each element of I^{**}). Then $\sum_{\eta \in H} \eta$ is in (13b). $\square$

According to the arguments in Section 3.17 of Geyer (2009) we can use our results for Poisson models to determine GDOR for multinomial and product multinomial models. Following this line we depart from the way our analysis went in preceding sections, presenting an algorithm for determining GDOR for product multinomial sampling. Instead we will use Algorithm 1.

Theorem 14. *Consider a Poisson GLM having response vector Y, mean value parameter vector μ, canonical parameter vector θ given by $\theta_i = \log(\mu_i)$, $i \in I$, and model matrix M, and let $\mathcal{A}$ be a partition of I. Suppose the indicator vectors u_A, $A \in \mathcal{A}$, having components $u_A(i)$ given by*

$$u_A(i) = \begin{cases} 1, & i \in A \\ 0 & i \in I \setminus A \end{cases} \tag{14}$$

are all in the column space of M. Then the MLE mean value parameter vector of this Poisson GLM, if it exists, is equal to the MLE mean value parameter vector for the product multinomial model obtained by conditioning as described in Theorem 11.

Moreover, the (possibly unique) MLE canonical parameter vector for this Poisson GLM, if it exists, is also a (nonunique) MLE canonical parameter vector for this product multinomial model, when the canonical parameterization for the product multinomial model is obtained from the canonical parameterization of the Poisson model by conditioning as described in Theorem 11 and the same model matrix and offset vector are used for both models.

When the MLE does not exist in the Poisson GLM, any GDOR for this model is also a GDOR for the product multinomial model, when the canonical parameterization for the product multinomial model is as described above.

Proof. The assertions about MLE follow from the "observed equals expected" property of maximum likelihood in a regular full exponential family.

Define I^{**} to be the set of all $i \in I$ such that there exists a δ satisfying the conditions of Theorem 1 and $\eta_i < 0$ where $\eta = M\delta$. Then by Theorems 1 and 2 a vector δ is a GDOR for the Poisson GLM if and only if for $\eta = M\delta$ we have $\eta_i < 0$ for $i \in I^{**}$ and $\eta_i = 0$ for $i \notin I^{**}$.

It is clear by Theorem 13 that any such GDOR for the Poisson problem is also a GDOR for the multinomial problem if the sets I^{**} defined by Theorems 2 and 13 are the same. From theorems 1 and 12 it is clear that every DOR for the Poisson problem is also a DOR for the multinomial problem. Hence the set I^{**} for the Poisson problem is a subset of the set I^{**} for the multinomial problem.

Conversely, suppose δ is a DOR for the multinomial problem and $\eta = M\delta$. To say that u_A, $A \in \mathcal{A}$ are in the column space of M is to say that each u_A is a linear combination of the columns of M, or, what is equivalent, that there exist δ_A, $A \in \mathcal{A}$ such that $u_A = M\delta_A$. Define ζ_A, $A \in \mathcal{A}$ by (11) and

$$\delta_{\text{pois}} = \delta - \sum_{A \in \mathcal{A}} \zeta_A \delta_A$$

and

$$\eta_{\text{pois}} = M\delta_{\text{pois}} = \eta - \sum_{A \in \mathcal{A}} \zeta_A u_A$$

Then for each $A \in \mathcal{A}$ we have

$$\max_{i \in A} \eta_{\text{pois}} = \zeta_A - \zeta_A = 0$$

So η_{pois} is a DOR for the Poisson problem. And η_{pois} has negative components that are the same as the components of η that were less than ζ_A for the A that contains the index of that component. This shows set I^{**} for the multinomial problem is a subset of the set I^{**} for the Poisson problem. $\square$

One might ask, since logistic regression is a special case of product multinomial sampling, why Algorithm 2? Couldn't we use Algorithm 1 instead, as we did for product multinomial? The answer is, yes we could, but

- Doing so doubles the size of the linear programming problem. Algorithm 1 will have n more variables and n more constraints than Algorithm 2, where n is the number of cases (row dimension of the model matrix).

- Doing so complicates the understanding of GDOR as we saw in Theorems 6 and 7, but only slightly. We still have $\eta_i \neq 0$ for a GDOR if and only if $Y_i = y_i$ almost surely in the LCM.

So, conversely, one might ask, if we use Algorithm 2 especially for product multinomial when multinomial is binomial, why don't we also use a special algorithm for general multinomial?

- Doing so increases the size of the linear programming problem. Algorithm 1 will have n more variables and n more constraints than needed to express the set of DOR for the multinomial, where n is the number of strata.

  But this cost is less than for binomial, because the strata are larger, hence fewer.

- Doing so complicates the understanding of GDOR as we saw in Theorems 12 and 13. It would similarly complicate any algorithm developed specifically for the product multinomial case.

- It is not even clear there is an algorithm analogous to algorithms 1 and 2 for the multinomial case.

  Of course, there are algorithms. Geyer (1990, Algorithms R and P) and Geyer (2009, Sections 3.12 and 3.13) give algorithms. But those algorithms are already known to be a lot slower than Algorithm 1. Eck and Geyer (2021, Supplementary Material) give algorithms. But those algorithms are already known to be more numerically unstable than Algorithm 1 (they need to compute the null space of the Fisher information matrix, a numerically difficult task, whereas Algorithm 1 makes its decisions based on whether the optimal value of the linear programming problem is zero or less than or equal to minus one).

  If we try to invent a new algorithm especially for the multinomial case that works analogously to Algorithm 1 we have the difficulty of trying to minimize $\eta_i - \max_{j \in A} \eta_j$ for each i. But max is not a linear operation. $\max_{j \in A} \eta_j$ is a convex function of η, so this takes us into general nonsmooth convex programming. Of course, such problems can be converted into linear programming problems by the method of slack variables. But this increases the size of the problem. Following this idea would perhaps take us to a linear program of the size of Algorithm 1 anyway. Since Algorithm 1 has such a nice natural interpretation when applied to multinomial sampling, we prefer it.

8.5 Numerical Stability

Since we are discussing computer programs, we have to consider whether the computer can approach the results described by our theorems. Our theorems use real real numbers. The computer does not. Its default computer arithmetic is inexact, having less than 16 significant figures (decimal) in its representation of what it calls real numbers.

This is not a problem for the computations recommended by Geyer (2009) which can use the exact rational arithmetic in R package `rcdd` (Geyer, et al., 2023) and thereby make no errors due to inexact computer arithmetic. But those algorithms are very slow on non-toy problems.

We could also use R function `lpcdd` in R package `rcdd` to solve linear programs. But it is slow and does not scale, getting far too slow for large problems.

We need something faster, hence Algorithm 1, which in this package uses R package `glpkAPI` (Gelius-Dietrich, 2022), which uses the default inexact computer arithmetic. Hence the concern about stability of solutions of linear programming.

It is simply a sad fact of life that computing with inexact computer arithmetic is not guaranteed to give correct answers. This is true of linear programming just as much as with any other area of computing that uses so-called floating point computer arithmetic.

At least we are using linear programs such that the answers should be clear for most problems. Our decision as to whether the optimal value is zero or not can use very sloppy tolerances because the optimal value is either zero or less than or equal to minus one.

The same goes for our decisions about what indices go in the set I^{**} that is the set of indices for which components of $\eta = M\delta$ for GDOR δ are nonzero. If we use a very sloppy tolerance for those decisions, then we will only make incorrect decisions in the direction of making I^{**} smaller. But this is OK in the middle of Algorithm 1 or 2 because any index that does not get put in I^{**} in one iteration of the loop will be reconsidered in other iterations.

When the algorithm stops because the optimal value of the linear program it does in some iteration is zero, the algorithm has tried to make some component of η with index in $I^* \setminus I^{**}$ less than or equal to minus one, and could not. So we can hopefully assume that it could not make them negative. So we are again, in a sense, using very sloppy tolerance.

8.6 Proofs

However, we do not actually care about the results of Algorithm 1 or 2 and certainly not that they were produced by linear programming. All we care about is whether or not there exists a δ such that $\eta = M\delta$ has the correct signs for its components.

How can we check that? We have the results: γ the putative GDOR and I^{**}, the index set of the components of the response vector that the LCM conditions on. Can we easily check that with exact rational arithmetic?

8.6.1 Proof of the First Kind

R package `gmp` (Lucas, et al., 2024) has methods for R generic function `solve` that use infinite precision rational arithmetic or modular integer arithmetic. Can we use that to get an exact construction of the GDOR?

We suggest the following problem. Find the vector x closest to γ such that Mx has the zero components it is supposed to have. Let Z be the matrix, whose rows are the rows of M indexed by $i \in I \setminus I^{**}$. We then solve the following optimization problem

$$\begin{aligned} &\text{minimize } \tfrac{1}{2}(x-\gamma)^T(x-\gamma) \\ &\text{subject to } Zx = 0 \end{aligned} \tag{15}$$

The Lagrangian function for this problem is

$$\mathcal{L} = \tfrac{1}{2}(x-\gamma)^T(x-\gamma) + \lambda^T Z x$$

where λ is a vector of Lagrange multipliers. The Kuhn-Tucker conditions are then

$$\begin{aligned} x - \gamma + Z^T\lambda &= 0 \\ Zx &= 0 \end{aligned} \tag{16}$$

This is a set of simultaneous linear equations to solve for x and λ. If it yields a solution such that Mx has the correct signs, then we have a mathematical proof that x is a DOR.

Note that there is nothing to be done for this kind of proof if Algorithm 1 or 2 stops in the first iteration with the result $\gamma = 0$ and $I^{**} = \varnothing$.

Note that there is also nothing to be done for this kind of proof if Algorithm 1 or 2 stops with $I^{**} = I$ in which case all components of η are far from zero, so we can presumably trust inexact computer arithmetic. But we can turn this observation into a formal mathematical proof by calculating $\eta = M\delta$ by infinite precision rational arithmetic and checking the signs of the components of η.

8.6.2 Proof of the Second Kind

In order to prove that Algorithm 1 or 2 stopped correctly if it did stop with optimal value zero (rather than $I^{***} = \varnothing$), we need to do redo that linear program with exact infinite precision rational arithmetic. R function `lpcdd` in R package `rcdd` can do this.

This proof, if successful, shows that the DOR found by the proof of the first kind is a GDOR. Hence it shows the MLE in the LCM does exist and is the MLE in the Barndorff-Nielsen completion of the OM.

In Section B below we see an example where the check of the linear program is trivial because the gradient of the objective function is the zero vector, hence the optimal value can only be zero. And we know there always is an optimal value because the zero vector is always feasible. If we want to check for this as a special case, we can.

Note that there is nothing to be done for this kind of proof if Algorithm 1 or 2 stops with $I^{***} = \varnothing$ in which case $I^{**} = I^*$. In this case, the DOR found in the preceding proof is automatically a GDOR.

8.6.3 Summary of Proof Theory

In the following,

- "the algorithm" means Algorithm 1 or 2 as the case may be,
- "nearly zero" means equal to zero according to the tolerance specified for the algorithm, and
- "use rational arithmetic" means
  - for matrix multiplication use R function `qmatmult` in R package `rcdd` or R function `%*%.bigq` in R package `gmp`, whichever is convenient,
  - for solving linear equations use R function `solve` in R package `gmp`, and
  - for linear programming use R function `lpcdd` in R package `rcdd`, and
- "η has the correct signs" means satisfies the conditions of Theorem 1 or 6, as the case may be.

If the algorithm stops with $\gamma = 0$ and `proofs = 2` was requested, then redo the linear program the algorithm did using rational arithmetic. If the optimal value is zero, then report `proofs = 2`.

If the algorithm stops with no component of $\eta = M\gamma$ nearly equal to zero, then compute $\eta = M\gamma$ using rational arithmetic (regardless of what level of proof was requested). If η has the correct signs, then report `proofs = 2`.

If the algorithm stops with $\gamma \neq 0$ and some component of $\eta = M\gamma$ nearly equal to zero and `proofs >= 1` was requested, then solve the linear equations (16) and calculate $\eta = Mx$, both using rational arithmetic. If η has correct signs, then report `proofs = 1` if `proofs = 1` was requested, else redo the last linear program the algorithm did using rational arithmetic. If the optimal value is zero, report `proofs = 2`.

In all other cases report `proofs = 0`.

8.6.4 Proof Certificates

As checkable proofs need to be, we need to return "certificates" of the proofs. For level one proofs, this would be the rational arithmetic version of the GDOR. One can then easily check that $\eta = M\delta$ has correct signs. For level two proofs, this would be the rational arithmetic primal and dual solutions of the linear program. The primal solution is a DOC because the optimal value is zero. The dual solution is the vector of Lagrange multipliers. One can then (more or less) easily check that the final linear program has a solution and the optimal value is zero, as in Section 5 of the vignette for R package `rcdd`.

This seems pretty esoteric for almost all users. The linear program is a bit hard to set up. One could, of course, look at the source code for R function `lpcdd`. But if one trusts that it is right, then one can also trust when it returns `proofs = 2`. And if one does not trust that source code, then one has to do the code check oneself. Perhaps we need a section on the help page for R function `lpcdd` about this. And an example.

8.6.5 Proofs Need Exact Data

All of this assumes we have the model matrix in some exact form, either integer-valued or rational-valued. But R function `sparse.model.matrix`, which we are using to turn formulas into model matrices does not do rational arithmetic. So, unless the model matrix happens to be integer-valued, we are in trouble anyway.

Consider the logistic regression example with quasi-complete separation of Agresti (2013, Section 6.5.1) that we analyze below (Section B). Agresti does not say whether we are supposed to treat the predictor variable as integer-valued or not. If we can take it to be integer-valued, and if we realize that we have two components of the response vector with the same predictor value, then we can rearrange the data so the predictor values are unique.

```
x <- seq(10, 90, 10)
success <- as.numeric(x >= 50)
failure <- as.numeric(x <= 50)
y <- cbind(success, failure)
data.frame(x, y)

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

Now the data say that one success and one failure are associated with the *same* predictor value ($x = 50$).

There is, when the data are given in this form, no trouble with inaccuracy of computer arithmetic. It is easily proven that the GDOR found in Section B is actually a GDOR.

But when the data are given in the form actually used in Section B the problem becomes insoluble. Then we do not know that one success and one failure are associated with the predictor value $x = 50$ if those values came from some previous calculation that might have involved inexact computer arithmetic.

Suppose the two predictor values that we entered as 50 were instead $50-\varepsilon$ and $50+\varepsilon$ for some small ε. Then the problem is either an example of complete separation or quasi-complete separation depending on which predictor value is associated with the success and which with the failure. And if we cannot know which (because we suspect inaccuracy of computer arithmetic is the cause), then there is no way to achieve mathematical certainty about what the LCM is.

Thus if our regression function is to follow the usual R pattern in which the model is specified by a formula and data in a data frame with no indication of how accurate the data are, then the problem of provably correct GDOR is insoluble. Even if we were to allow some optional way of indicating accuracy of data, naive users would not use it.

As we know, naive users do not read the documentation, and those naive users who keep asking question 7.31 of the R FAQ (Hornik, 2020) have never even encountered the idea that computer arithmetic is inexact. If our R package has to deal with users like that, then we can never know whether the answers it computes are correct. We have to apply Algorithms 1 and 2 and hope.

8.6.6 Guidance About Proofs

For less naive readers we can provide the guidance to make the model matrix integer-valued if they can. This means making quantitative covariates (class `"numeric"`) integer-valued. Dummy variables are, of course, zero-or-one valued, so qualitative variables (class `"character"` or class `"factor"`) present no problems.

We can also give advice to avoid duplicate predictor values if possible, but that advice may be hard to follow in complicated applications.

8.7 Limiting Conditional Models

Theorem 15. *For Poisson sampling, if the set I^{**} defined in Theorem 2 is nonempty, then the LCM conditions on the event*

$$Y_i = y_i, \qquad i \in I^{**}. \tag{17}$$

For binomial sampling, if the set I^{**} defined in Theorem 7 is nonempty, then the LCM conditions on the event (17). For multinomial or product multinomial sampling with partition $\mathcal{A}$, if the set I^{**} defined in Theorem 13 is nonempty, then the LCM conditions on the event (17). But, if we define

$$I^{**+} = \bigcup \{\, A \in \mathcal{A} : A \setminus I^{**} \text{ is a singleton set} \,\},$$

then the LCM also conditions on the event

$$Y_i = y_i, \qquad i \in I^{**} \cup I^{**+}.$$

Proof. If δ is a GDOR and $\eta = M\delta$, then the LCM conditions on the event $\langle Y - y, \eta \rangle = 0$, and in all cases this is equivalent to (17). In the multinomial or product multinomial case if for some $A \in \mathcal{A}$ the event (17) conditions $y_i = 0$ for all but one $i \in A$, then it also conditions $y_j = n_A$ for the $j \in A$ such that $y_j \neq 0$. ☐

9 Reporting Multinomial Model Fits

Theorem 14 already tells us we can use the results of fitting a Poisson GLM or detecting whether a GDOR exists for a Poisson GLM to fit or detect whether a GDOR exists for a multinomial or product multinomial LLM.

An issue remains. The regression coefficients for the regressors u_A defined by (14) are not identifiable because each u_A is a direction of constancy of the saturated model. Thus we do not report these coefficients. They would not be coefficients if we were fitting the correct model (multinomial or product multinomial rather than Poisson).

And another issue remains. For the regression coefficients that we do report, standard errors, Fisher information, and inverse Fisher information should be based on the multinomial or product multinomial distribution rather than the Poisson distribution.

Fisher information for the saturated model canonical parameter vector is, of course,

$$I(\theta) = \operatorname{var}(Y)$$

and for product multinomial sampling with strata set $\mathcal{A}$ the i, j component is

$$\begin{aligned} \operatorname{var}(Y_i) &= n_A \pi_i (1 - \pi_i), && i \in A \in \mathcal{A}, i = j \\ \operatorname{cov}(Y_i, Y_j) &= -n_A \pi_i \pi_j, && i \in A \in \mathcal{A}, j \in A \\ \operatorname{cov}(Y_i, Y_j) &= 0, && i \in A \in \mathcal{A}, j \notin A \end{aligned}$$

We see that the Fisher information matrix is block diagonal with blocks being strata and off-diagonal terms not in the same block being equal to zero. We can get this block structure with what is called `modmat.strata` in the examples in the appendix. Here we denote this matrix M_s. Its columns say which components of the response vector are in a stratum. Thus $M_s M_s^T$ is a matrix of the same shape as the Fisher information matrix that says which components of the Fisher information matrix are nonzero.

We also have to deal with different strata having different sample sizes, but we can use (8) for that. If `mu` is the mean value parameter vector and `pi` is the usual parameter vector, whose components satisfy (8) then

```
(Diagonal(x = mu) - tcrossprod(mu, pi)) * tcrossprod(modmat.strata)
```

computes the Fisher information matrix for the saturated model canonical parameter vector (this uses R functions in R package `Matrix`).

Then the Fisher information matrix for the submodel canonical parameter vector is

$$I(\beta) = M^T I(\theta) M$$

where M is the model matrix for the submodel (for the coefficients we keep in the submodel, not the ones we throw away as explained above).

10 Table-Valued Response?

No. Just no. Although we find the idea of fitting contingency tables as contingency tables, as R function `loglin` does and as R function `loglm` in R package `MASS` optionally does, elegant, it is just too confusing when combined with solutions at infinity.

The way these functions work is to use the iterative proportional fitting (IPF) algorithm to operate on mean values only. They never use canonical parameters in any way. This allows them to estimate solutions on the boundary correctly (at infinity for canonical parameters is on the boundary for mean value parameters), in the sense that the cells of the contingency table that have mean value zero in the LCM converge to zero (slowly) as IPF continues to iterate. IPF also deals with structural zeros correctly if given a starting position that has the correct structural zeros (it never changes a zero cell to nonzero).

But if one wants more than point estimates, if one wants to do valid hypothesis tests and confidence intervals following Geyer (2009), then one needs GDOR, canonical parameter estimates for the LCM, and so forth. So one needs to ignore what R function `loglin` or `loglm` does and start over with the methods described in this document.

It is easy to convert a contingency table to a data frame

```
data(exercise_6.28)
foo <- xtabs(counts ~ ., data = exercise_6.28)
class(foo)

## [1] "xtabs" "table"

dim(foo)

## [1] 2 2 2 3 2

bar <- as.data.frame(foo)
identical(exercise_6.28, bar)

## [1] FALSE

names(bar)

## [1] "Occupational_aspirations"
## [2] "Socioeconomic_status"
## [3] "IQ"
## [4] "Residence"
## [5] "Gender"
## [6] "Freq"

for (i in levels(bar$Occupational_aspirations))
    for (j in levels(bar$Socioeconomic_status))
        for (k in levels(bar$IQ))
            for (l in levels(bar$Residence))
                for (m in levels(bar$Gender))
                    stopifnot(with(exercise_6.28,
                       counts[Occupational_aspirations == i &
                       Socioeconomic_status == j &  IQ == k &
                       Residence == l & Gender == m]) ==
                       with(bar, Freq[Occupational_aspirations == i &
                       Socioeconomic_status == j &  IQ == k &
                       Residence == l & Gender == m]))
```

So it is a bit annoying that this procedure re-orders the data, but since any order is equivalent, this doesn't matter.

Thus users can easily go from tabular or array data to dataframe data using R function `as.data.frame`.

Thus users can easily use the methods already described that want data as a dataframe rather than as a table or array.

11 Vector, Matrix, or Factor Response

For binomial or multinomial family, we need to figure out the order in which to deal with the possibilities. There is a natural order, although it was not apparent before doing the examples in the appendix of this document (to your humble author).

If the response `y` is a factor, convert it to a matrix with the command

```
y <- sparse.model.matrix(~ 0 + y)
```

but save the original response factor for putting in the output (perhaps).

If `family == "binomial"` then it is easy to convert this matrix to a response vector `y` and a sample size vector `n` with the commands

```r
n <- rowSums(y)
y <- y[ , 1]
```

but save the response matrix for putting in the output (perhaps).

If `family == "multinomial"` then it is more complicated to convert this matrix to response and strata vectors. But the following commands do the job

```r
strata <- paste0("row", row(y))
y <- as.vector(y)
```

but save the response matrix for putting in the output (perhaps).

And now we are done except in the case where the response was originally a vector. If `family == "binomial"` then we have to create a sample size vector `n` to be in tune with what we get when the response comes in other forms.

```r
n <- rep(1, length(y))
```

If `family == "multinomial"` then we have to create a strata model matrix and a covariates model matrix. Both are tricky.

In the case that `strata` was missing we create the strata model matrix with the command

```r
modmat.strata <- model.matrix(~ 1)
```

In the case that `strata` was specified as a factor we create the strata model matrix with the command

```r
modmat.strata <- sparse.model.matrix(~ 0 + strata)
```

(this includes the case where the response was supplied as a matrix and we created strata as a character vector, which will be treated like a factor by R function `sparse.model.matrix`)

In the case that `strata` was specified as a formula `f` we create the strata model matrix with the command

```r
modmat.strata.formula <- sparse.model.matrix(strata, data)
modmat.strata.formula <- as.matrix(modmat.strata.formula)
strata <- apply(modmat.strata.formula, 1, paste0)
strata <- as.factor(strata)
```

Now for the model matrix for covariates. In the case that the response was specified originally as a vector, we just make this model matrix in the usual way applying R function `sparse.model.matrix` to the formula.

But in the case that the response was specified originally as a matrix or factor (by now converted to a matrix) we have to back up, or at least use that matrix (which we have saved rather than clobbered). We make a model matrix (not the final one) by applying R function `sparse.model.matrix` to the formula. Then we make a list, all of whose components are the same, this non-final model matrix. Then we do something to change the column names of the matrices in this list so they are all distinct. Then we run this list through R function `bdiag` making a block diagonal matrix, and that is the final model matrix for covariates. All of this is illustrated in Section D.2 in the appendix.

12 Hypothesis Tests

As usual for R packages for regression-like analysis, R generic function `anova` does hypothesis tests, and we write methods for it handling objects of class `"llmdr"` output by R function `llmdr`.

The fundamental theory of hypothesis testing when the MLE does not exist in the OM is given by Section 3.15 in Geyer (2009), where it is attributed to Stephen Fienberg (who outlined this theory in answer to a question from your humble author during a talk).

Firstly, the hypothesis test involves (as always) only the distribution under the null hypothesis (we are not discussing power calculation here, and even if we were, they only involve local alternatives near the null hypothesis). Hence the only computational geometry (solutions as infinity) involves the null hypothesis.

If the MLE exists for the OM for the null hypothesis, then we could do the hypothesis test using R functions `glm` and `anova`, ignoring any warnings because we know this actually does the right thing despite the warnings.

If the MLE does not exist for the OM for the null hypothesis, then we need the theory in Geyer (2009) (attributed to Fienberg). This says we use the same conditioning for both models, null and alternative, and this conditioning is that for the LCM for the null hypothesis. We pay no attention to directions of recession, solutions at infinity, and so forth for the alternative hypothesis. This means we have to recalculate the degrees of freedom for the alternative hypothesis. It is not necessarily either of the degrees of freedom in the output of R function `llmdr`. Those are the degrees of freedom for the alternative hypothesis if the MLE exists in the OM for the null hypothesis and the degrees of freedom for the alternative hypothesis if it were (contrary to fact) treated as the null hypothesis.

The degrees of freedom for the alternative hypothesis is a bit complicated. Let η be the GDOR for the saturated model for the null hypothesis, and let M be the model matrix for the alternative hypothesis. then, if the family is not multinomial, the degrees of freedom is the dimension of the column space of $M[\eta = 0, \ ]$ in R notation, that is, the part of M consisting of the rows for which the components of η are zero.

For multinomial sampling, the degrees of freedom recalculation is even more complicated. Let M_s be the model matrix for strata for the alternative hypothesis, then we consider both $M[\eta = 0, \ ]$ and $M_s[\eta = 0, \ ]$. Let V denote the column space of the former, and V_s denote the column space of the latter. The degrees of freedom for the alternative hypothesis is the dimension of the vector $V \cap V_s^\perp$.

This last recipe is so complicated, that R function `anova.llmdr` passes the buck to R function `glm.fit`. It fits the multinomial response model having model matrix M, model matrix for strata M_s, and subset vector $\eta == 0$. And following Theorem 14 we fit this model using R function `glm.fit` giving it the model matrix

```r
cbind(modmat.strata, modmat)[eta == 0, , drop = FALSE]
```

in R notation, where `modmat`, `modmat.strata`, and `eta` are M, M_s, and η in the notation above. The `coefficients` component of the result is what R function `glm.fit` thinks is MLE for this model, and that vector has `NA` for components that are not identifiable. But we know that none of the components that corresponded to M_s are identifiable for the corresponding product multinomial model. Hence the degrees of freedom are given by the R code

```r
beta <- gout$coefficients
# remove components for strata
beta <- beta[- seq(1, ncol(modmat.strata))]
df.alt <- sum(! is.na(beta))
```

where `gout` is the result returned by R function `glm.fit` and `df.alt` is the degrees of freedom for this model when considered as an alternative hypothesis and the null hypothesis has GDOR η for the saturated model canonical parameterization. (This `beta` is what R function `llmdr` would report for the `coefficients` vector for this model if (contrary to fact) `eta` were the GDOR for this model rather than for the null hypothesis.)

12.1 Likelihood Ratio Tests

(Also called the Wilks test.) The maximized value of the log likelihood is the same when done by R function `llmdr` as when done by R functions `glm` or `multinom` (the latter in R package `nnet`), modulo inexactness of computer arithmetic.

Thus we take the deviances computed by R function `llmdr` and the degrees of freedom discussed above (recalculating the degrees of freedom for the alternative hypothesis), and do the test in the usual way except for calculating the degrees of freedom differently than from what other R functions would do.

12.2 Rao Tests

(Also called score tests, also called Lagrange multiplier tests.) These were not mentioned in Geyer (2009) but have been an option for R function `anova` since R version 2.14.0 (released in October, 2011). So we should have them too.

Theoretically, there is no issue. Rao tests only depend on the MLE for the null hypothesis. They do not use anything about the alternative hypothesis except its model matrix. For this section, just call this matrix M_{alt}. Then the score that gives the score test its name is

$$M_{\text{alt}}^T(y - \hat{\mu})$$

where y is the response vector and $\hat{\mu}$ is its MLE expected value under the null hypothesis. This would, of course, be the zero vector if we replaced M_{alt} by the analogous model matrix for the null hypothesis. This vector has asymptotic variance matrix

$$M_{\text{alt}} I_{\text{null}}(\hat{\theta}) M_{\text{alt}}^T$$

where the middle factor is the Fisher information matrix for the saturated model canonical parameter (also called linear predictor) for the null hypothesis. The exact variance matrix is, of course,

$$M_{\text{alt}} I_{\text{null}}(\theta) M_{\text{alt}}^T$$

where θ is the true unknown parameter value (assumed to be in the null hypothesis). Thus the Rao test statistic is

$$(y - \hat{\mu})^T M_{\text{alt}}^T \bigl(M_{\text{alt}} I_{\text{null}}(\hat{\theta}) M_{\text{alt}}^T\bigr)^{-1} M_{\text{alt}} (y - \hat{\mu})$$

The degrees of freedom are, of course, the same as for the likelihood ratio test.

12.3 Reporting Multiple Tests

R function `anova` traditionally does multiple tests when given a sequence of models, testing each adjacent pair of models.

When solutions at infinity are involved, a model has two different degrees of freedom: one considered as a null hypothesis and one considered as an alternative. So we have to report both. This complicates the traditional version of an ANOVA table, giving it an extra column.

12.4 Reporting Multiple Tests for One Single Model Fit

R function `anova` traditionally does a bunch of tests when given one model object. We don't allow this.

One gets more or less the same featurality from R function `drop1` if we implement that.

12.5 R Functions Add1 and Drop1

R users expect R functions `add1` and `drop1` to work with objects returned by model fitting functions like `llmdr`, that is, although users don't necessarily think of it this way, there must be methods of these functions for objects of class `"llmdr"`.

Looking at the source code for R functions `add1.glm` and `drop1.glm` one finds that they produce the following:

- for `drop1`, vectors of indices of columns of the model matrix of the given model that give the submodels of the given model resulting from the "drops"

- and, for `add1`, a model matrix for a model containing all terms to add and vector of indices of columns of this new model matrix that give the supermodels of the given model resulting from the "adds".

Thus we need a version of R function `llmdr` that works with model matrices explicitly.

Also experience shows that there are models one wants to fit that cannot be expressed in the R fofmula mini-language. So such a function is needed anyway. It should be documented and exported by the package. We could call this function `llmdr.fit` analogous to `glm.fit` but we could also make R function `llmdr` generic and give it methods for objects of class `formula` to handle formulas and a default method to handle model matrices. (This is what R package `aster` does with its model-fitting function)

But with `llmdr` we have the issue that not all families need the same sorts of objects to specify them. To review

- for Poisson family, we just need the response vector and the model matrix,
- for binomial family, we just need the the model matrix and the response vector, matrix, or factor as the case may be, and
- for multinomial family, we just need the the model matrix and the response vector, matrix, or factor as the case may be and we may also need strata specified by a factor or a formula.

So we need a proposal about how to specify this extra stuff.

For the response, we propose providing it in the same form for both methods, that is, for binomial or multinomial family, it can be vector, matrix, or factor. And the same form has the same meaning for both methods.

The number of rows of the model matrix and the response matrix are the same (if the response is provided as a matrix). If the response is a vector or a factor, then the length of the response is the number of rows of the model matrix.

We need a specification of strata only if the family is multinomial and the response is a vector. Even then we allow strata to be missing, which is taken to mean that there is exactly one stratum (so the sampling scheme is multinomial rather than product multinomial).

For the `formula` method of R generic function `llmdr` the strata can be specified by a factor or a formula. For the default method, since we don't allow formulas to specify the model matrix, then perhaps we shouldn't allow them to specify the strata either. We propose that strata should be specified by another model matrix whose elements are zero-or-one-valued and whose columns indicate the strata. Let y denote the response vector and M_s this model matrix for strata (this same concept of model matrix for strata and the same notation M_s was used in Sections 9 and 12 above). This model matrix for strata is also returned by R function `llmdr` as component `modmat.strata` of the object it returns. So users can get this matrix from any previous call of R function `llmdr` applied to the same sampling scheme for the same data.

13 Confidence Regions and Intervals

13.1 Our Proposal

We are going to start by slightly modifying the proposal of Geyer (2009), Section 3.16.2. Let H denote these support of the LCM, which is described by Theorem 15. Conditioning on the event $Y \in H$ turns the OM into the LCM.

Then a confidence region for how close the true unknown parameter value is to infinity in the canonical parameter space or how close it is to the boundary in the mean value parameter space is the set

$$\{\, \beta : \operatorname{pr}_\beta(Y \in H) \geq \alpha \,\} \tag{18}$$

where the desired coverage probability is $1 - \alpha$ (or any other parameterization can be substituted for β).

This is a confidence region that completely ignores the data for the LCM. It just says how close in the OM we are likely to be to the LCM. Now for any function $g(\beta)$ (or of any other parameter, we can form a confidence interval by minimizing and maximizing $g(\beta)$ over (18). All of these intervals have simultaneous coverage because they are based on the same confidence region (18).

13.2 Conventional Confidence Regions for LCM

Another confidence region is more conventional. We make a likelihood-based confidence region for the parameters of the LCM. If l_{LCM} is the log likelihood for the LCM, then this region is

$$\{\, \beta : l_{\text{LCM}}(\beta) \geq l_{\text{LCM}}(\hat{\beta}) - \text{crit} \,\} \tag{19}$$

where β is the same parameter as in (18), the parameter of the OM, and crit is an appropriate chi-square critical value that uses the degrees of freedom (number of parameters estimated) of the LCM.

We should make it perfectly clear that the β in (19) is the full vector β, the submodel canonical parameter of the OM. It is not the β that is estimated in the LCM with components dropped to make the LCM have identifiable canonical parameterization. Thus this β is not an identifiable parameter of the LCM (usually we make sure it is an identifiable parameter of the OM), and this means the region (19) is also unbounded in certain directions, those that are directions of constancy of the LCM.

Neither of the regions (18) or (19) are much good by themselves. They tell us something but not everything. In general, we have to combine them by some correction for multiple testing to get all the information.

But there are two special cases where we only want one of these confidence regions and not the other. If the MLE exists in the OM so the LCM is the OM, then we only want the conventional confidence region (19), which in this case is a bounded region. When we use only this, we get conventional results. More on this later. The second special case is when the LCM is a completely degenerate distribution, which says the only value the response vector could have is the one that was observed. In this case the LCM has no identifiable parameters and (19) just gives the whole vector space where β lives. Since it does nothing, we should ignore it and only use (18).

Note that we can write l_{LCM} as

$$l_{\text{LCM}}(\beta) = \log \operatorname{pr}_\beta(y \mid Y \in H)$$

from which we see that (18) only depends on a marginal distribution and (19) only depends on the corresponding conditional distribution. Alternatively, (18) only depends on the components of the response vector that are fixed by the conditioning event $Y \in H$, and (19) only depends on the rest of the components of the response vector (that are unaffected by the conditioning event $Y \in H$).

The probabilities multiply (joint equals marginal times conditional) so we should be able to get a better correction for simultaneous coverage than Bonferroni.

Further discussion of the theory of these confidence intervals is deferred to Section 13.11 below.

13.3 Correction for Two Confidence Regions

If α_1 is the probability that (18) fails to cover and α_2 is the probability that (19) fails to cover, then the probability that they both cover is

$$(1 - \alpha_1)(1 - \alpha_2) = 1 - \alpha_1 + \alpha_2 + \alpha_1\alpha_2$$

whereas Bonferroni would give the probability that either fails to cover is less than $\alpha_1 + \alpha_2$ hence the probability that they both cover is at least

$$1 - \alpha_1 + \alpha_2$$

A check with Wikipedia (2021) shows this method of correcting for multiple comparisons is well known when the data for hypothesis tests or confidence intervals are independent (they do not mention joint equals marginal times conditional), but they do not give a name for this procedure.

So our method of multiplying probabilities using joint equals marginal times conditional does do better than Bonferroni. So it seems that we had better change the name of the argument of R function `confint.llmdr` from what we guessed would make sense at first: `bonferroni`. This has now been done.

Anyway, that function has arguments to determine $1-\alpha_1$ and $1-\alpha_2$ that are `simultaneous = TRUE` (more about this later), `bonferroni = c(1, 1)` which was supposed to mean $\alpha_1 = \alpha_2$ and in general that the proportions of $1 - \alpha_1$ and $1 - \alpha_2$ are what is specified, and `level = 0.95`, the desired overall confidence level, $(1 - \alpha_1)(1 - \alpha_2)$. So this gives us two equations to solve for $1 - \alpha_1$ and $1 - \alpha_2$

$$\begin{aligned} \frac{\texttt{level1}}{\texttt{level2}} &= \frac{\texttt{bonferroni[1]}}{\texttt{bonferroni[2]}} \\ \texttt{level1} * \texttt{level2} &= \texttt{level} \end{aligned}$$

where $\texttt{level1} = 1 - \alpha_1$ and $\texttt{level2} = 1 - \alpha_2$. Solving for `level1` gives

$$\frac{\texttt{bonferroni[1]}}{\texttt{bonferroni[2]}} * \texttt{level2} = \texttt{level1} = \frac{\texttt{level}}{\texttt{level2}}$$

so

$$\begin{aligned} \texttt{level2} &= \sqrt{\texttt{level} * \frac{\texttt{bonferroni[2]}}{\texttt{bonferroni[1]}}} \\ \texttt{level1} &= \sqrt{\texttt{level} * \frac{\texttt{bonferroni[1]}}{\texttt{bonferroni[2]}}} \end{aligned}$$

except we need to think of another name for `bonferroni`.

Note that these equations can make `level1` and `level2` outside the range of $(0, 1)$, in which case we must mean that one of these is equal to `level` and the other to one (meaning we don't use that kind of region, the only way one gets 100% confidence is to make no restriction). Also note that we cannot have both components of `bonferroni` equal to zero because then we get `0 / 0 = NaN`.

13.4 Simultaneous Coverage or Not

Now for the intended meaning of `simultaneous = FALSE`. Geyer (2009) tries to provide something analogous to the confidence intervals provided by most R functions, which are not corrected for simultaneous coverage, although it snarkily says those are "something users *think* they can interpret" (emphasis added), meaning users actually misinterpret these (all users, always). Now there is just no way that the confidence region (18) can be made into non-simultaneous intervals for different parameters. It is inherently simultaneous. Being close to the boundary for mean value parameters or close to infinity for canonical parameters is not a function of one parameter alone but of all of them together. But there is a way to make non-simultaneous confidence intervals based on (19): just use one degree of freedom in calculating the chi-square critical value rather than the number of identifiable parameters of the LCM. In hindsight, we now think we might as well just get the benefit of simultaneous coverage, except perhaps when the MLE exists in the OM. Or except perhaps when the number of identifiable parameters of the LCM is large.

The argument above does not agree with Geyer (2009). There our confidence region (18) is made smaller in an attempt to make it nonsimultaneous. We no longer think this is a good idea. Hence the proposal in this document.

13.5 Argument Override

Argument `override = TRUE` (the default) means use confidence region (18) by itself when the LCM is completely degenerate and use confidence region (19) by itself when the MLE exists in the OM. And `override = FALSE` would say that even in these cases we should use `level1` and `level2` even though that wastes some of our coverage probability. The only reason we are even providing this option is because we don't feel we completely understand this stuff, although we understand it a lot better than we did in 2009.

13.6 Summary

So this gives us an algorithm.

- Determine `level1` and `level2` as discussed based on arguments `level`, `bonferroni`, `override`, and whether the LCM is completely degenerate, LCM $=$ OM, or otherwise.

- determine α_1 and `crit` using the argument `simultaneous`, as discussed in determining `crit`

- For each function $g(\beta)$ that determines a parameter we want to estimate solve the optimization problem

$$\begin{aligned} &\text{minimize or maximize } g(\beta) \\ &\qquad\text{subject to } \operatorname{pr}_\beta(Y \in H) \geq \alpha_1 \\ &\qquad\qquad l_{\text{LCM}}(\beta) \geq l_{\text{LCM}}(\hat{\beta}) - \text{crit} \end{aligned}$$

For one-sided confidence intervals, the maximum or minimum will be at infinity or minus infinity for canonical parameters and on the boundary (equal to the MLE) for mean value parameters, and our code should recognize this situation and give the appropriate answer without invoking some optimization algorithm which will have no clue about how to find an answer at infinity. The GDOR helps here. We know that β goes to infinity in the GDOR direction.

For these nonlinear optimization problems we have had good results from R function `auglag` in R package `alabama` (Varadhan, 2023).

13.7 R Generic Function Predict?

One last consideration: our function `confint.llmdr` almost completely replaces R generic function `predict`, which is misnamed anyway (a referee for Geyer, et al. (2007) complained that it doesn't make "predictions" except for linear models, it just provides parameter estimates and standard errors) and the author of R package aster (Geyer, 2023) agreed but replied that R generic function `predict` is the function that does this job, misnamed or no, and we have to use it.

Now we are thinking that maybe we don't need a method for R generic function `predict` for objects of class `"llmdr"`. Maybe the `confint` method does everything wanted? Not quite.

- Methods for R generic function `predict` usually have a `newdata` argument that allows predictions for hypothetical individuals not in the observed data. So unless we add a `newdata` argument to our method for R generic function `confint`, we cannot replace R generic function `predict`.

- But the way `predict` usually works, providing estimates and (if requested) standard errors, won't work for us, because standard errors are meaningless when the MLE is in the LCM (they would all be infinite).

- Also in R package `aster` (Geyer, 2023) the method of R generic function `predict` for aster models provides not only estimates of all the kinds of parameters that our method of R generic function `confint` for LLMDR objects does but also provides estimates and (if requested) standard errors for any linear functions of them. Since our methods of producing confidence intervals do not involve normal approximation and the delta method, we can allow confidence intervals for arbitrary functions of any of the parameters.

  The only issue with allowing arbitrary functions (arbitrary g in our notation above) is that the optimizer may return a local optimum that is not a global optimum, and thus give the wrong endpoint of the confidence interval.

But we punt on all of this. Leave it for future versions of the package.

13.8 More on Constraint Functions

Theorem 16. *For Poisson sampling, the probability in (18) has log*

$$q_1(\beta) = - \sum_{i \in I^{**}} \mu_i \tag{20a}$$

where, as usual, $\theta = a + M\beta$ and $\mu = \exp(\theta)$, the exp *function operating componentwise, as in R, and I^{**} is computed by Algorithm 1. And the log likelihood in (19) is*

$$q_2(\beta) = \sum_{i \in I \setminus I^{**}} (y_i \theta_i - \mu_i) \tag{20b}$$

For binomial sampling, the probability in (18) has log

$$q_1(\beta) = \sum_{i \in I^{**}} \bigl[y_i \log(\pi_i) + (n_i - y_i) \log(1 - \pi_i) \bigr] \tag{20c}$$

where, as usual, $\theta = a + M\beta$ and $\pi = \operatorname{logit}^{-1}(\theta)$, the inverse logit *function also operating componentwise, as in R, and I^{**} is computed by Algorithm 2. And the log likelihood in (19) is*

$$q_2(\beta) = \sum_{i \in I \setminus I^{**}} \bigl[y_i \log(\pi_i) + (n_i - y_i) \log(1 - \pi_i) \bigr] \tag{20d}$$

For product multinomial sampling with strata $\mathcal{A}$, the probability in (18) has log

$$q_1(\beta) = \sum_{A \in \mathcal{A}} n_A \log \left(\sum_{i \in A \setminus I^{**}} \pi_i \right) \tag{20e}$$

where, as usual, $\theta = a + M\beta$ and

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

and I^{**} is defined in Theorem 13. And the log likelihood in (19) is

$$q_2(\beta) = \sum_{A \in \mathcal{A}} \sum_{A \setminus I^{**}} y_i \log(\pi_i) \tag{20f}$$

Proof. For (20a) the probability of the data being in the support of the LCM is

$$\prod_{i \in I^{**}} \frac{\mu_i^{y_i}}{y_i!} e^{-\mu_i}$$

but $y_i = 0$ for all $i \in I^{**}$ so this simplifies to

$$\prod_{i \in I^{**}} e^{-\mu_i}$$

and taking logs gives (20a). For (20b) we have the usual Poisson log likelihood except we only sum over components of the response vector that are free (not conditioned) in the LCM.

For (20c) the probability of the data being in the support of the LCM is

$$\prod_{i \in I^{**}} \binom{n_i}{y_i} \pi_i^{y_i} (1 - \pi_i)^{n_i - y_i}$$

but $y_i = 0$ or $y_i = n_i$ for all $i \in I^{**}$ so this simplifies to

$$\prod_{i \in I^{**}} \pi_i^{y_i} (1 - \pi_i)^{n_i - y_i}$$

and taking logs gives (20c). For (20d) we have the usual binomial log likelihood except we only sum over components of the response vector that are free (not conditioned) in the LCM.

For (20e) the PDF of the product multinomial is

$$\prod_{A \in \mathcal{A}} (n_A!) \prod_{i \in A} \frac{\pi_i^{y_i}}{y_i!}$$

For the event $Y \in H$ we have $y_i = 0$ for all $i \in I^{**}$. So the PDF simplifies to

$$\prod_{A \in \mathcal{A}} (n_A!) \prod_{i \in A \setminus I^{**}} \frac{\pi_i^{y_i}}{y_i!}$$

Now all of the remaining variables are free in the LCM and can take any nonnegative integer values subject to sums over strata being equal to the sample sizes. So to calculate the probability of the event $Y \in H$ we sum over all of those possible values and obtain

$$\sum_{A \in A} \left(\sum_{i \in A \setminus I^{**}} \pi_i \right)^{n_A}$$

from the multinomial theorem (the reason we don't get one is because the probabilities in the inner sums do not sum to one because we are not summing over the whole stratum. Taking logs gives (20e). For (20f) we have the usual product multinomial log likelihood except we only sum over components of the response vector that are free (not conditioned to be equal to zero) in the LCM. □

We notice that (20c) and (20d) have the same formula summed over disjoint index sets. There is a difference that is hidden in the notation. In (20c) we know that every y_i is either 0 or n_i, and in (20d) we don't know that.

With this in mind, we notice that (20a) and (20b) also have the same formula summed over disjoint index sets, even though it does not look like that. But if we rewrite (20a) as

$$q_1(\beta) = \sum_{i \in I^{**}} (y_i \theta_i - \mu_i)$$

we know that $y_i = 0$ for $i \in I^{**}$ so this simplifies to (20a).

Thus for both Poisson and binomial sampling q_1 and q_2 are likelihoods of certain exponential families, hence concave. But we have no such analgous property for product multinomial sampling. Exponential families with dependence among components of the response are just different. The function (20e) is not the log likelihood of any full exponential family. It is the log likelihood of the missing data exponential family where we are missing the values of the response vector for $i \notin I^{**}$, but that is not helpful, because exponential families with missing data don't have any nice properties over and above the usual asymptotics of maximum likelihood. In particular, they don't have concave log likelihoods.

For all of the formulas in the theorem that are log likelihoods, the first derivatives are trivial: observed minus expected, like for all regular full exponential families. But that leaves us with one formula we don't know what the derivative is.

Theorem 17. *The derivative of* (20e) *is*

$$\frac{\partial q_1(\beta)}{\partial \theta_j} = \sum_{\substack{A \in \mathcal{A} \\ j \in A}} \frac{1}{\sum_{i \in A \setminus I^{**}} \pi_i} \sum_{k \in A \setminus I^{**}} \frac{\partial \mu_k}{\partial \theta_j} \tag{21}$$

This equation needs some interpretation. The leftmost sum always has exactly one term. Because $\mathcal{A}$ is a partition, there is always exactly one $A \in \mathcal{A}$ such that $j \in A$. The sum in the denominator is never empty because $A \setminus I^{**}$ is never empty (we cannot have all components of a multinomial random vector equal to zero, they must sum to the sample size). The partial derivatives $\partial \mu_k / \partial \theta_j$ are components of the Fisher information matrix for θ. They are discussed in Section 9 above.

Proof. By the chain rule

$$\frac{\partial q_1(\beta)}{\partial \theta_j} = \sum_{k \in I} \frac{\partial q_1(\beta)}{\partial \pi_k} \frac{\partial \pi_k}{\partial \theta_j}$$

so

$$\begin{aligned}
\frac{\partial q_1(\beta)}{\partial \theta_j} &= \sum_{A \in \mathcal{A}} \sum_{k \in A} \frac{\partial q_1(\beta)}{\partial \pi_k} \frac{\partial \pi_k}{\partial \theta_j} \\
&= \sum_{A \in \mathcal{A}} \sum_{k \in A \setminus I^{**}} \frac{n_A}{\sum_{i \in A \setminus I^{**}} \pi_i} \cdot \frac{\partial \pi_k}{\partial \theta_j} \\
&= \sum_{A \in \mathcal{A}} \frac{n_A}{\sum_{i \in A \setminus I^{**}} \pi_i} \sum_{k \in A \setminus I^{**}} \frac{\partial \pi_k}{\partial \theta_j} \\
&= \sum_{\substack{A \in \mathcal{A} \\ j \in A}} \frac{n_A}{\sum_{i \in A \setminus I^{**}} \pi_i} \sum_{k \in A \setminus I^{**}} \frac{\partial \pi_k}{\partial \theta_j}
\end{aligned}$$

and the last equality is the fact that $\partial \pi_k / \partial \theta_j = 0$ unless k and j are in the same stratum. Finally we use $n_A \pi_k = \mu_k$ when $k \in A$. □

13.9 Random Sampling?

In doing confidence intervals for the first example in the appendix (Section A.6 below) we hit upon the idea (more out of frustration than anything else) of sampling a bunch of points in the confidence region for β to serve as starting points for the optimization algorithms. Since we have many non-convex optimization problems to do, many random starting points can help find the correct local optima (the ones that are global optima), especially if they are fairly densly distributed throughout the feasible region, the intersection of the two constraint sets (18) and (19).

There is no need for the points to have any particular probability distribution. We just need a fair amount of the spread throughout the region.

Since the region may be infinite, this gives problems. Also distances in β space have little to do with distances between probability distributions. So we hit on the idea of having τ uniformly distributed rather than β. This would seem to have a more physical meaning and have the probability distributions themselves closer to uniformly distributed.

When we transform $\tau \to \beta$, the Jacobian in the change-of-variable theorem is the determinant of the derivative of the inverse transformation $\beta \to \tau$, which is the Fisher information matrix.

For binomial and product multinomial sampling the mean value parameter space is a bounded region, hence this probability distribution exists.

For Poisson sampling, the mean value parameter space is an unbounded region, hence we need a theorem about that.

Theorem 18. *For Poisson sampling, the intersection of* (18) *and* (19) *is bounded when mapped $\beta \to \tau$.*

Proof. If $\mu_i \to \infty$ for some $i \in I^{**}$, then (20a) goes to zero. Rewrite (20b) as

$$q_2(\beta) = \sum_{i \in I \setminus I^{**}} [y_i \log(\mu_i) - \mu_i]$$

Then it is clear that, If $\mu_i \to \infty$ for some $i \in I \setminus I^{**}$, then (20b) goes to zero. Thus all means must stay bounded in order to stay in both of these confidence regions. □

We naturally turn to MCMC to sample this distribution, but since we really don't care about the distribution, only about getting a bunch of well spread out points, we can be very sloppy about the MCMC.

We probably don't need MCMC at all.

13.10 Dimension of the LCM Canonical Parameter

When making the "conventional" confidence region for the submodel canonical parameter β, we need to be careful about dimension. We need to use only identifiable parameters of the OM. Perhaps the best way to figure this out is just to punt it to R function `glm`. Fit the original model, ignoring any warnings R function `glm` emits (because we don't care what it thinks about solutions at infinity). But do pay attention to which components of the `coefficients` vector it makes `NA`. If `gout` is the object returned by R function `glm`, then `! is.na(gout$coefficients)` indicates the parameters involved in the confidence regions (18) and (19).

Except that won't work for multinomial because R function `glm` doesn't do multinomial, except when we are being clever, as described by Theorem 14. So fit the Poisson model for the OM corresponding to the product multinomial as described in the theorem. Then eliminate the parameters for strata. then look at which of the remaining coefficients are reported `NA`.

13.11 More on Our Theory of Confidence Regions

A theory about a confidence region proposal must discuss all possible data values. The confidence region is valid if it has the desired coverage probability. For discrete data there cannot be exact non-randomized hypothesis tests or confidence regions (Geyer and Meeden, 2005). Thus our procedure can have (at best) only approximately correct coverage probability (averaged over all possible data values).

13.11.1 One-Parameter Models

To fix some ideas, we first do one-parameter models. For a concrete example, we do the binomial distribution, for which Geyer (2021b) shows our proposal works. The coverage probability is, of course

$$\sum_{y=0}^{n} \operatorname{pr}_\pi(y) I(\text{interval for } y \text{ covers } \pi)$$

where in this section $I(\,\cdot\,)$ is the function that maps logical values false and true to numerical values zero and one (respectively).

We are assuming the conventional confidence intervals work when π is far from the boundary of its parameter space. Hence this is approximately $1 - \alpha$ for such π.

So what is the probability the interval for 0 covers π? By definition it fails to cover when $\text{pr}_\pi(Y = 0)$ is less than α. So the probability it covers is greater than or equal to $1 - \alpha$ when π is in the region.

This argument confuses even your humble author, so let be even more specific in our calculations. $\text{pr}_\pi(Y = 0) = (1 - \pi)^n$ where n is the sample size, so the interval is the set of π satisfying

$$(1 - \pi)^n \geq \alpha$$

or

$$1 - \pi \geq \alpha^{1/n}$$

or

$$\pi \leq 1 - \alpha^{1/n}$$

So let us consider a few cases

Thus we have the argument

- When π is such that $\text{pr}_\pi(Y = 0) \geq \alpha$, the interval covers π with probability at least $1 - \alpha$ because the the interval for $y = 0$ does that all by itself (never mind other values of y).
- When π is such that $\text{pr}_\pi(Y = n) \geq \alpha$, the interval covers π with probability at least $1 - \alpha$ because the the interval for $y = n$ does that all by itself (never mind other values of y).
- Otherwise, we are assuming conventional confidence intervals work (approximately), so we have coverage $1 - \alpha$ (approximately).

Thus we see that our proposal is (in this case) a conservative correction to conventional confidence intervals. They guarantee correct coverage when the observed data is on the boundary and otherwise do no worse than conventional confidence intervals.

13.11.2 Multi-Parameter Models

So how does this case splitting work for multi-parameter models? For this we need to introduce the notion of the convex support of an exponential family. This is the smallest closed convex set that contains the canonical statistic vector with probability one. This is the set called C_{sub} in Geyer (2009).

Any convex set is partitioned partioned by the family of relative interiors of its nonempty faces (Rockafellar, 1970, Theorem 18.2). The support of the

LCM is the unique face of the convex support that contains the observed value of the canonical statistic vector in its relative interior (Geyer, 2009, Section 3.8; this was also known by Barndorff-Nielsen and Brown).

The conditioning event $M^T Y \in H_{\text{sub}}$ where

$$H_{\text{sub}} = \{ \, z \in \mathbb{R}^K : \langle z - M^T y, \delta \rangle = 0 \, \} \tag{22}$$

(Geyer, 2009, Section 3.14.2), can be rewritten as $M^T Y \in F$ where $F = C_{\text{sub}} \cap H_{\text{sub}}$ is the smallest face of C_{sub} containing $M^T y$ in its relative interior (when δ is the GDOR for observed data y or when $\delta = 0$ when $M^T y$ is in the relative interior of C_{sub}) (Geyer, 2009, Section 3.8).

Let $\mathcal{F}$ denote the family of nonempty faces of C_{sub} (this includes C_{sub}, which is always a face of itself, by definition). Let $\mathcal{E}$ be the subset of $\mathcal{F}$ consisting of zero-dimensional faces (extreme points). These are the faces for which (when the `override` argument of R function `confint.llmdr` is `TRUE`, Section 13.5 above) for which we only use the proposal in Section 13.1 above and not the one in Section 13.2 above).

The coverage probability is, of course

$$\sum_{y \in S} \text{pr}_\beta(y) I(\text{region for } y \text{ covers } \beta)$$

where S is the sample space of the model. And we divide this up

$$\begin{aligned} \sum_{F \in \mathcal{F}} \sum_{\substack{y \in S \\ M^T y \in \operatorname{rint} F}} & \text{pr}_\beta(y) I(\text{region for } y \text{ covers } \beta) \\ &= \sum_{F \in \mathcal{F}} \sum_{\substack{y \in S \\ M^T y \in \operatorname{rint} F}} \text{pr}_\beta(y) I(q_{1,F}(\beta) > c_{1,F}) I(q_{2,y}(\beta) > c_{2,F}) \end{aligned}$$

where we have changed notation from Theorem 16 writing $q_{1,F}$ instead of q_1 to emphasize that this function depends on the data but only through the event $M^T y \in \operatorname{rint} F$, that is, on which F is the convex support of the LCM, whereas q_2 depends on y, not all components of the response vector, but rather those that are free (not constrained by the conditioning on $M^T y \in F$) in the LCM, and where we have introduced critical values $c_{1,F}$ and $c_{2,F}$ which also depend on F (because they depend on whether the LCM is completly degenerate, partially degenerate, or equal to the OM).

So now we examine cases.

- In case $F \in \mathcal{E}$ and $M^T y \in \operatorname{rint} F$, this is the case whre the LCM is completely degenerate, $\operatorname{rint} F = F$ is the set containing only the single point $M^T y$, where y is the observed value of the response vector, we have $I(q_{2,y}(\beta) > c_{2,F}) = 1$ regardless of the value of β. And we have coverage at least $1 - \alpha$ for all β satisfying $q_{1,F}(\beta) > c_{1,F}$ by the same argument as in the one-dimensional case.

- In case $F \in \mathcal{F}$ and $M^T y \in \operatorname{rint} F$ and $F \notin \mathcal{E}$ and $F \neq C_{\text{sub}}$ this is the case where the LCM is partially degenerate, and the argument becomes more complicated. Still we have for all β satisfying $q_{1,F}(\beta) > c_{1,F}$ and $q_{2,y}(\beta) > c_{2,F}$

  $$\operatorname{pr}_\beta(M^T Y \in F) \geq \alpha_1 \tag{23a}$$

  $$l_{\text{LCM},F}(\beta) \geq l_{\text{LCM},F}(\hat{\beta}) - c_2 \tag{23b}$$

  (where α_1 and c_2 do not depend on F for F in this case).

  It looks like we have to do this by mathematical induction. Fix $G \in F$ such that $G \neq C_{\text{sub}}$, and suppose (this is the induction hypothesis) that for all $F \in \mathcal{F}$ such that $F \subset G$ and $F \neq G$ we have coverage at least $1 - \alpha$ for all β satisfying (23a) and (23b).

REVISED DOWN TO HERE

Then the probability that our confidence region proposal covers a particular true unknown parameter value β is

$$\sum_{F \in \mathcal{F}} \operatorname{pr}_\beta(Y \in \operatorname{rint} F) I(q_{1,F}(\beta) > c_1) \\ \times E_\beta \left\{ \operatorname{pr}_\beta(q_{2,Y}(\beta) \geq c_2 \mid Y \in F) \,\middle|\, Y \in \operatorname{rint} F \right\} \tag{24}$$

for some constants c_1 and c_2, and where $I(\,\cdot\,)$ maps false and true to 0 and 1, and where we have changed notation from Theorem 16 writing $q_{1,F}$ instead of q_1 to emphasize that this function depends on the data only throught the event $Y \in \operatorname{rint} F$, that is, on which F is the convex support of the LCM, whereas q_2 depends on Y, not all components of the response vector, but rather those that are free (not constrained by the conditioning on $Y \in F$) in the LCM.

Now we choose c_2 so the conditional expectation in the formula above is greater than $1 - \alpha_2$, approximately, assuming the data for the LCM has large enough "sample size" so the usual asymptotics of maximum likelihood work for the LCM. This reduces our coverage probability calculation to (approximately)

$$(1 - \alpha_2) \sum_{F \in \mathcal{F}} \operatorname{pr}_\beta(Y \in \operatorname{rint} F) I(q_{1,F}(\beta) > c_1)$$

Now

$$q_{1,F}(\beta) = \log \operatorname{pr}_\beta(Y \in F)$$

There is a mismatch in our theory here. It would make for a cleaner analysis if we substituted $\operatorname{rint} F$ for F here. But calculation of $\operatorname{rint} F$ involves calculating all of the faces of F, which is a very complicated problem in computational geometry. The whole point of the line of research of Geyer (2009), Eck and Geyer (2021), and this design document is to avoid having to do that, because it can take hours, days, weeks or worse of computing time for even moderately large problems. None of the examples in Geyer (2009) much less the big data example in Eck and Geyer (2021) could have been done if we insisted on calculating $\operatorname{rint} F$.

So now our coverage probability becomes (approximately)

$$(1 - \alpha_2) \operatorname{pr}_\beta \left(\bigcup \{ \operatorname{rint} F : F \in \mathcal{F} \text{ and } q_{1,F}(\beta) \geq c_1 \} \right) \tag{25}$$

assuming we are using `override = FALSE` in the arguments to R function `confint` (described in Section 13.5 above). If we are using `override = TRUE`, then this becomes more complicated. Let $\mathcal{E}$ be the subset of $\mathcal{F}$ consisting of zero-dimensional faces (extreme points). Then our coverage probability becomes (approximately)

$$\begin{aligned}(1 - \alpha) \left(\operatorname{pr}_\beta(\operatorname{rint} C) + \sum_{F \in \mathcal{E}} \operatorname{pr}_\beta(Y \in \operatorname{rint} F) \right) \\ + (1 - \alpha_2) \sum_{F \in (\mathcal{F} \setminus \mathcal{E}) \setminus \{C\}} \operatorname{pr}_\beta(Y \in \operatorname{rint} F) I(q_{1,F}(\beta) > c_1)\end{aligned} \tag{26}$$

So somehow we have to argue here that this probability is approximately $1 - \alpha$. And we want to do this without having to calculate F for any face of C except for the convex support of the LCM.

14 Information Criteria

14.1 Kullback-Leibler Information

14.2 Takeuchi Information Criterion

The Takeuchi information criterion (TIC) is

$$\text{TIC} = -2l(\hat{\theta}) + 2\,\text{tr}\big[I(\theta_0)J(\theta_0)^{-1}\big]$$

where l is the log likelihood function and

$$\begin{aligned} I(\theta) &= \text{var}\left\{\nabla l(\theta)\right\} \\ J(\theta) &= E\left\{-\nabla^2 l(\theta)\right\} \end{aligned}$$

where expectations are taken with respect to the true unknown distribution of the data (which may not be any model under consideration) and where θ_0 is the θ that maximizes

$$E\left\{l(\theta)\right\} \tag{27}$$

assuming this maximizer exists and is unique (this depends on the true unknown distribution and the statistical model, so existence and uniqueness cannot be proved) and also satisfies

$$E\left\{\nabla l(\theta_0)\right\} = 0$$

which will hold assuming differentiation under the integral sigh is valid in (27).

This is a general bias-corrected estimator of twice Kullback-Leibler information assuming the usual regularity conditions for misspecified maximum likelihood (Konishi and Kitagawa, 2008, Sections 3.3.5 and 3.4.3; Burnham and Anderson, 2002, Sections 2.3 and 7.3).

14.3 Akaike Information Criterion

The Akaike information criterion (AIC) is derived from TIC by assuming the model for which we are calculating AIC is correct (contains the true unknown distribution), in which case θ_0 is the true unknown parameter value and $I(\theta_0) = J(\theta_0)$ is the Fisher information matrix. Then the penalty term is twice the trace of the identity matrix, which is the number of identifiable parameters of the model.

Now we should ask what this has to do with solutions at infinity. If a model under consideration is correct in the sense that the true unknown distribution lies not in that model but rather in its Barndorff-Nielsen completion, then the argument above in not correct.

The "usual asymptotics of maximum likelihood" do not apply to the OM but rather to the LCM containing the true unknown distribution. So AIC should use the number of identifiable parameters of that LCM, perhaps zero in the case of a completely degenerate LCM.

This much is clear. The question is whether anyone will like this conclusion. After all, AIC is not used under the assumption that all models are correct. If they were, one would just choose the most parsimonious without looking at AIC. So assume each model is correct is used only to get the penalty $2p$ and then forgotten (un-assumed). When we compare models using AIC, we are not assuming all of them are correct (even though that assumption is used in the derivation).

One way to think of what AIC does is that it says this is an unbiased estimate of twice Kullback-Leibler information if that model is correct, and if it is not correct the bias will be worse, so this is a criterion that makes this model as good as it possibly can (without cheating and doing no bias correction at all). If that is the argument, it is still clear that p should be the number of identifiable parameters of the LCM (perhaps zero).

14.4 Bayesian Information Criterion

15 Empty Models and Other Anomalies

15.1 Empty Models

We need to decide what to do with empty models, things like

```
y <- rbinom(10, 1, 0.5)
modmat <- sparse.model.matrix(y ~ 0)
dim(modmat)

## [1] 10  0
```

As the book *Effective Java* says (a book with more about how to write R or any other computer language than most books on R) in item 54 "Return Empty Collections or Arrays, Not Nulls" so in R this means never return `NULL` unless you always return `NULL`.

If the user is expecting a numeric vector, return `numeric(0)` rather than `NULL`. And similarly for other return types.

Let's see what R function `glm` does.

```
gout <- glm(y ~ 0, family = "binomial")
class(gout)

## [1] "glm" "lm"

names(gout)

##  [1] "coefficients"      "residuals"
##  [3] "fitted.values"     "effects"
##  [5] "R"                 "rank"
##  [7] "qr"                "family"
##  [9] "linear.predictors" "deviance"
## [11] "aic"               "null.deviance"
## [13] "iter"              "weights"
## [15] "prior.weights"     "df.residual"
## [17] "df.null"           "y"
## [19] "converged"         "boundary"
## [21] "model"             "call"
## [23] "formula"           "terms"
## [25] "data"              "offset"
## [27] "control"           "method"
## [29] "contrasts"         "xlevels"
```

Just what is expected. We should do the same. But

```
coef(gout)

## numeric(0)
```

So we can see that Bloch's recommendation is the R way (at least here).

15.2 Empty Strata

By definition the set of strata should partition the index set of the response vector. We need to check for both conditions.

- No stratum is empty.
- The union of the strata is the index set of the respose vector.

A Complete Separation Example of Agresti

A.1 Data

Agresti (2013, Section 6.5.1) discusses the following toy problem, which is a simple (one predictor) logistic regression.

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

A.2 First Linear Program

So we use Algorithm 2.

```
modmat <- sparse.model.matrix(y ~ x)

tangent.direction <- as.numeric(y == 0) - as.numeric(y == 1)

objgrd <- rbind(tangent.direction) %*% modmat
objgrd <- as(objgrd, "numeric")

lp <- initProbGLPK()
addRowsGLPK(lp, nrow(modmat))

## [1] 1

addColsGLPK(lp, ncol(modmat))

## [1] 1

setSimplexParmGLPK(MSG_LEV, GLP_MSG_OFF) # STFU
# row bounds
lower.bounds <- pmin(- tangent.direction, 0)
upper.bounds <- pmax(- tangent.direction, 0)
idx <- which(lower.bounds == upper.bounds)
setRowsBndsGLPK(lp, idx, lower.bounds[idx],
    upper.bounds[idx], rep(GLP_FX, length(idx)))
idx <- which(lower.bounds != upper.bounds)
setRowsBndsGLPK(lp, idx, lower.bounds[idx],
    upper.bounds[idx], rep(GLP_DB, length(idx)))
# col bounds: free
idx <- 1:ncol(modmat)
setColsBndsGLPK(lp, idx, rep(-Inf, length(idx)),
    rep(Inf, length(idx)), rep(GLP_FR, length(idx)))
# objective function
setObjDirGLPK(lp, GLP_MIN)
idx <- which(objgrd != 0)
setObjCoefsGLPK(lp, idx, objgrd[idx])
# constraint matrix (modmat)
foompter <- mat2triplet(modmat)
loadMatrixGLPK(lp, nnzero(modmat), foompter$i, foompter$j, foompter$x)

# do it
solveSimplexGLPK(lp)

## [1] 0

# have solution?
getPrimStatGLPK(lp) == GLP_FEAS

## [1] TRUE

getObjValGLPK(lp)

## [1] -5
```

Because the optimal value is negative, the MLE does not exist in the OM, and the solution is a DOR.

```
delta <- getColsPrimGLPK(lp)
eta <- getRowsPrimGLPK(lp)
names(delta) <- colnames(modmat)
names(eta) <- x

delta

## (Intercept)           x
##      -1.250       0.025

eta
```

```
##    10    20    30    40    60    70    80    90
## -1.00 -0.75 -0.50 -0.25  0.25  0.50  0.75  1.00

all(eta * tangent.direction <= 0)

## [1] TRUE

all(tangent.direction != 0 | eta == 0)

## [1] TRUE
```

Since all components of `eta` are nonzero, this says that `delta` is a GDOR and the LCM is completely degenerate, concentrated at the observed data. There is no need to do another linear program because we have discovered $I^{**} = I$. We are exiting the loop in Algorithm 2 at the $I^{***} = \varnothing$ test.

A.3 Limiting Conditional Model

There is nothing to do to fit the LCM, since it is completely degenerate, what Agresti calls complete separation.

But we do have something to report: the GDOR.

```
coefs <- cbind(NA, delta, NA, NA, NA)
rownames(coefs) <- colnames(modmat)
colnames(coefs) <- c("Estimate", "GDOR", "Std. Error",
    "z value", "Pr(>|z|)")
```

```
printCoefmat(coefs)

##             Estimate   GDOR Std. Error z value
## (Intercept)       NA -1.250         NA      NA
## x                 NA  0.025         NA      NA
##             Pr(>|z|)
## (Intercept)       NA
## x                 NA
```

This may not be very helpful to users. More useful is the information that the LCM is completely degenerate. It conditions all components of the response vector to be equal to their observed values.

A.4 Clean Up

```r
delProbGLPK(lp)
```

A.5 Proof

We follow the suggestion of Section 8.6.3 and give a rigorous proof that we have found a GDOR.

```r
modmat <- as(modmat, "matrix")
modmat <- gmp::as.bigq(modmat)
eta <- gmp::`%*%`(modmat, cbind(delta))
eta

## Big Rational ('bigq') 8 x 1 matrix:
##      [,1]
## [1,] -72057594037927935/72057594037927936
## [2,] -27021597764222975/36028797018963968
## [3,] -36028797018963965/72057594037927936
## [4,] -4503599627370495/18014398509481984
## [5,] 9007199254740995/36028797018963968
## [6,] 36028797018963975/72057594037927936
## [7,] 6755399441055745/9007199254740992
## [8,] 72057594037927945/72057594037927936

all(eta * tangent.direction <= 0)

## [1] TRUE

all(tangent.direction != 0 | eta == 0)

## [1] TRUE
```

Because η has the correct signs and no zero components, this proves η is a GDOR.

A.6 Confidence Intervals

A.6.1 Constraint Set

Since the LCM is completely degenerate, we only have intervals based on (18).

For that we need to encode the probability in that formula, and to avoid underflow and catastrophic cancellation in inexact computer arithmetic, we use log probability. Letting $\theta = a + M\beta$, as always, this log probability is

$$\sum_{\substack{i \in I \\ y_i = 0}} \log\left(\frac{1}{1 + e^{\theta_i}}\right) + \sum_{\substack{i \in I \\ y_i = 1}} \log\left(\frac{1}{1 + e^{-\theta_i}}\right)$$

or

$$- \sum_{\substack{i \in I \\ y_i = 0}} \operatorname{log1p}\bigl(e^{\theta_i}\bigr) - \sum_{\substack{i \in I \\ y_i = 1}} \operatorname{log1p}\bigl(e^{-\theta_i}\bigr) \tag{28}$$

where $\operatorname{log1p}(x) = \log(1+x)$ is a function provided by R and C/C++ in order to avoid catastrophic cancellation when x is small and the answer should be $\operatorname{log1p}(x) \approx x$ rather than `-Inf` when `1 + x == x` because `x` is smaller than the machine epsilon.

We can help the optimizer by giving it derivatives of objective and constraint functions. Call (28) $h(\beta)$, then

$$\begin{aligned}
\frac{\partial h(\beta)}{\partial \theta_k} &= -I(y_k = 0) \frac{e^{\theta_k}}{1 + e^{\theta_k}} - I(y_k = 1) \frac{-e^{-\theta_k}}{1 + e^{-\theta_k}} \\
&= -I(y_k = 0) \frac{e^{\theta_k}}{1 + e^{\theta_k}} + I(y_k = 1) \frac{1}{1 + e^{\theta_k}} \\
&= -I(y_k = 0)\pi_k + I(y_k = 1)(1 - \pi_k)
\end{aligned}$$

where, just for this equation and the next two $I(\,\cdot\,)$ is the function that maps `FALSE` to zero and `TRUE` to one. Then by the chain rule we have

$$\frac{\partial h(\beta)}{\partial \beta_l} = \sum_{k \in K} \bigl[-I(y_k = 0)\pi_k + I(y_k = 1)(1 - \pi_k)\bigr] m_{kl}$$

where m_{kl} are the components of the model matrix M.

A.6.2 For Theta

We are going to produce confidence intervals for components of the saturated model canonical parameter vector. We know that these confidence intervals are all one-sided in this problem. If $y_i = 0$, then we know the lower endpoint of the confidence interval is $-\infty$. If $y_i = 1$, then we know the upper endpoint of the confidence interval is $+\infty$.

So let's do it.

```
conf.level <- 0.95
alpha <- 1 - conf.level

class(modmat)

## [1] "bigq"

modmat <- gmp::asNumeric(modmat)
class(modmat)

## [1] "matrix" "array"

confun <- function(beta) {
    stopifnot(is.numeric(beta))
    stopifnot(is.finite(beta))
    stopifnot(length(beta) == ncol(modmat))
    theta <- drop(modmat %*% cbind(beta))
    return(- sum(log1p(exp(  theta[y == 0])))
           - sum(log1p(exp(- theta[y == 1]))) - log(alpha))
}

confun.jac <- function(beta) {
    stopifnot(is.numeric(beta))
    stopifnot(is.finite(beta))
    stopifnot(length(beta) == ncol(modmat))
    theta <- drop(modmat %*% cbind(beta))
    mu <- 1 / (1 + exp(- theta))
    return(rbind(- as.numeric(y == 0) * mu +
        as.numeric(y == 1) * (1 - mu)) %*% modmat)
}

lower <- rep(-Inf, length(y))
upper <- rep(Inf, length(y))

beta.start <- c(0, 0)
```

```
for (i in seq(along = y)) {

    objfun <- function(beta) {
        stopifnot(is.numeric(beta))
        stopifnot(is.finite(beta))
        stopifnot(length(beta) == ncol(modmat))
        theta <- drop(modmat %*% cbind(beta))
        return(theta[i])
    }

    objfun.gradient <- function(beta)
        modmat[i, ] # want drop = TRUE

    if (tangent.direction[i] > 0) {
        # need to find upper endpoint
        aout <- auglag(beta.start, objfun,
            objfun.gradient, confun, confun.jac,
            control.outer = list(trace = FALSE),
            control.optim = list(fnscale = -1))
        stopifnot(aout$convergence == 0)
        upper[i] <- aout$value
    } else if (tangent.direction[i] < 0) {
        # need to find lower endpoint
        aout <- auglag(beta.start, objfun,
            objfun.gradient, confun, confun.jac,
            control.outer = list(trace = FALSE))
        stopifnot(aout$convergence == 0)
        lower[i] <- aout$value
    }
}

cbind(lower, upper)

##           lower      upper
## [1,]       -Inf -0.9185667
## [2,]       -Inf -0.4303787
## [3,]       -Inf  0.2852351
## [4,]       -Inf  2.9441695
## [5,] -2.9438807        Inf
## [6,] -0.2852351        Inf
## [7,]  0.4303787        Inf
## [8,]  0.9185668        Inf
```

Seems to have worked!

A.6.3 For Pi

Let's look at a plot, transforming these to probabilities.

```
errbar(x, y, 1 / (1 + exp(- upper)), 1 / (1 + exp(- lower)),
    xlab = "x", ylab = "probability")
```

A.6.4 For Beta

We also want to try confidence intervals for β and τ to see how goofy they are. First β. From the fact that the GDOR is

```
delta

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

we see that β_1 goes all the way to $-\infty$ and β_2 goes all the way to $+\infty$. So we are looking for an upper bound on β_1 and a lower bound on β_2.

```
aout <- auglag(beta.start, function(beta) beta[1],
    function(beta) c(1, 0), confun, confun.jac,
    control.outer = list(trace = FALSE),
    control.optim = list(fnscale = -1))
stopifnot(aout$convergence == 0)
upper <- c(aout$value, Inf)
aout <- auglag(beta.start, function(beta) beta[2],
    function(beta) c(0, 1), confun, confun.jac,
    control.outer = list(trace = FALSE))
stopifnot(aout$convergence == 0)
lower <- c(-Inf, aout$value)
foo <- rbind(lower, upper)
colnames(foo) <- names(delta)
t(foo)
```

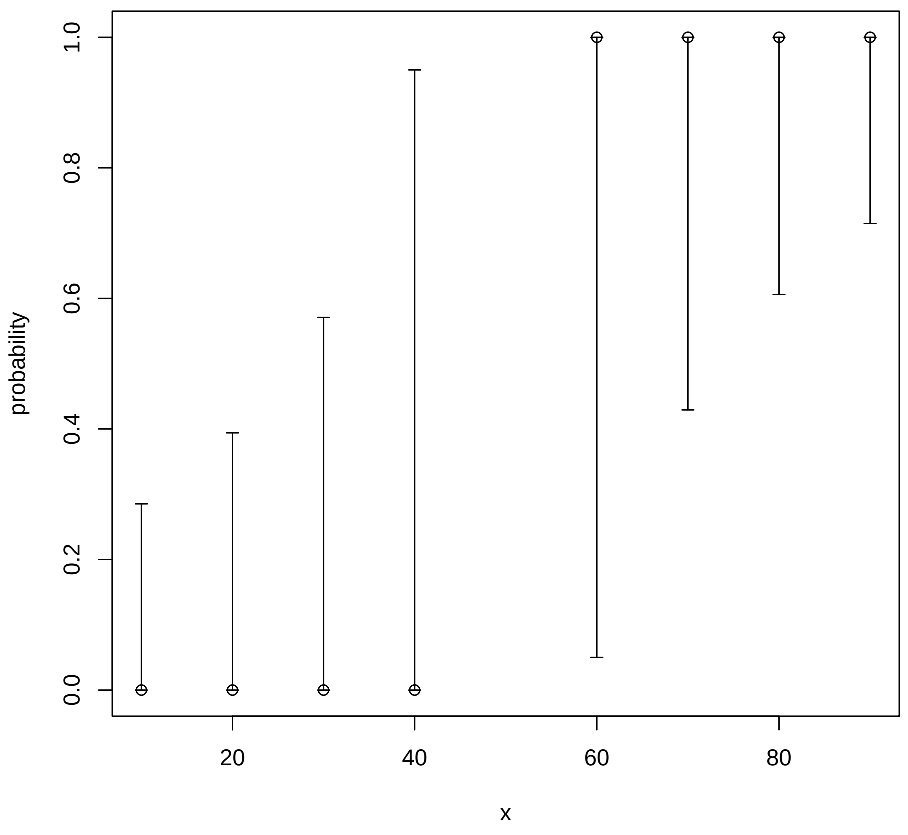

Figure 1: Confidence intervals for probabilities, hollow circles are MLE.

```
##                    lower    upper
## (Intercept)         -Inf -1.34039
## x             0.03348093      Inf
```

A.6.5 For Tau

Now for τ. The parameters τ and β are the same length but not in the same vector space. They are in dual vector spaces: τ lives in the same vector space as y, the sample space of the canonical statistic, and β lives in the dual space of that, the vector space containing the canonical parameter space. So the GDOR does not tell us which way τ goes to the boundary, it says when it is on the boundary: recall that $\langle Y - y, \delta \rangle \leq 0$ almost surely, so Y is close to the boundary when $\langle Y - y, \delta \rangle$ is small.

That says the confidence region for τ lies in a half space, but it doesn't say that confidence *intervals* are one-sided.

Now the MLE and canonical sufficient statistic are

```
tau.hat <- drop(crossprod(modmat, y))
names(tau.hat) <- names(delta)
tau.hat

## (Intercept)           x
##           4         300
```

We know from theory that the derivative of τ with respect to β is the Fisher information matrix for β, which is the variance of Y.

So let's try that

```
tau <- function(beta) {
    stopifnot(is.numeric(beta))
    stopifnot(is.finite(beta))
    stopifnot(length(beta) == ncol(modmat))
    theta <- drop(modmat %*% beta)
    mu <- 1 / (1 + exp(- theta))
    drop(crossprod(modmat, mu))
}

tau.jac <- function(beta) {
    stopifnot(is.numeric(beta))
    stopifnot(is.finite(beta))
    stopifnot(length(beta) == ncol(modmat))
    theta <- drop(modmat %*% beta)
    mu <- 1 / (1 + exp(- theta))
    fisher <- diag(mu * (1 - mu))
    t(modmat) %*% fisher %*% modmat
}

tau(beta.start)

## [1]   4 200

tau.jac(beta.start)

##      [,1] [,2]
## [1,]    2  100
## [2,]  100 6500

jacobian(tau, beta.start)

##      [,1] [,2]
## [1,]    2  100
## [2,]  100 6500
```

Looks good. So let's try intervals.

```
# tau[1] lower
aout <- auglag(beta.start, function(beta) tau(beta)[1],
    function(beta) tau.jac(beta)[1,], confun, confun.jac,
    control.outer = list(trace = FALSE))
stopifnot(aout$convergence == 0)
lower <- aout$value
# tau[1] upper
aout <- auglag(beta.start, function(beta) tau(beta)[1],
    function(beta) tau.jac(beta)[1,], confun, confun.jac,
    control.outer = list(trace = FALSE),
    control.optim = list(fnscale = -1))
stopifnot(aout$convergence == 0)
upper <- aout$value
# tau[2] lower
```

```
aout <- auglag(beta.start, function(beta) tau(beta)[2],
    function(beta) tau.jac(beta)[2,], confun, confun.jac,
    control.outer = list(trace = FALSE))
stopifnot(aout$convergence == 0)
lower <- c(lower, aout$value)
# tau[2] upper
aout <- auglag(beta.start, function(beta) tau(beta)[2],
    function(beta) tau.jac(beta)[2,], confun, confun.jac,
    control.outer = list(trace = FALSE),
    control.optim = list(fnscale = -1))
stopifnot(aout$convergence == 0)
upper <- c(upper, aout$value)

foo <- cbind(lower, upper)
rownames(foo) <- names(delta)
foo

##                  lower     upper
## (Intercept)    2.21829   5.78171
## x            171.64508 300.00000

tau.hat

## (Intercept)           x
##           4         300
```

From previous experience (Stat 5421 Notes) we know this is wrong. Both intervals must be two-sided. So either there is a bug in the code above or the optimizer got confused by some local optimum or something of the sort.

Let's try something else. Let us sample a random distribution of points uniformly distributed on the confidence region for τ (Section 13.9 above).

```
# Find a feasible starting point

beta.start <- c(0, 0)
for (i in 1:100) if (confun(beta.start + i * delta) > 0) break
i

## [1] 2

beta.start <- beta.start + i * delta
beta.start

## (Intercept)           x
##       -2.50        0.05

confun(beta.start)

## [1] 0.7643724

jeff <- function(beta) {
    if (confun(beta) <= 0) return(-Inf)
    fisher <- tau.jac(beta)
    as.vector(determinant(fisher)$modulus)
}

jeff(beta.start)

## [1] 7.011383

mout <- metrop(jeff, beta.start, nbatch = 1000,
    scale = 1 / delta)
mout$accept; mout$time

## [1] 0
##    user  system elapsed
##   0.019   0.000   0.019

mout <- metrop(mout, scale = 1e-2 / delta, nspac = 10)
mout$accept; mout$time

## [1] 0.0356
##    user  system elapsed
##   0.188   0.000   0.187

mout <- metrop(mout, scale = 1e-3 / delta)
mout$accept; mout$time

## [1] 0.3512
##    user  system elapsed
##   0.347   0.000   0.347
```

```
efficiency <- function(x) {
    foo <- initseq(x)
    foo$var.con / foo$gamma0
}
apply(mout$batch, 2, efficiency)

## [1] 229.612602   1.092284

mout <- metrop(mout, scale = 2e-3 / delta, nspac = 100)
mout$accept; mout$time

## [1] 0.1907
##    user  system elapsed
##   2.547   0.004   2.552

apply(mout$batch, 2, efficiency)

## [1] 234.353085   1.251074
```

Good enough. This gives crude confidence intervals for τ.

```
taus <- apply(mout$batch, 1, tau)
dim(taus)

## [1]    2 1000

foo <- apply(taus, 1, range)
rownames(foo) <- c("lower", "upper")
colnames(foo) <- names(delta)
t(foo)

##                 lower      upper
## (Intercept)   2.84174   5.774812
## x           192.19740 353.394234
```

Already we can see that the upper bound of the interval for τ_2 is higher than the MLE, which we did not have before.

Return to optimization, hoping we do better with better starting points.

```r
beta.start.save <- beta.start
i <- which(taus[,1] == min(taus[,1]))
beta.start <- mout$batch[i, ]
# tau[1] lower
aout <- auglag(beta.start, function(beta) tau(beta)[1],
    function(beta) tau.jac(beta)[1,], confun, confun.jac,
    control.outer = list(trace = FALSE))
stopifnot(aout$convergence == 0)
lower <- aout$value
# tau[1] upper
i <- which(taus[,1] == max(taus[,1]))
beta.start <- mout$batch[i, ]
aout <- auglag(beta.start, function(beta) tau(beta)[1],
    function(beta) tau.jac(beta)[1,], confun, confun.jac,
    control.outer = list(trace = FALSE),
    control.optim = list(fnscale = -1))
stopifnot(aout$convergence == 0)
upper <- aout$value
# tau[2] lower
i <- which(taus[,2] == min(taus[,2]))
beta.start <- mout$batch[i, ]
aout <- auglag(beta.start, function(beta) tau(beta)[2],
    function(beta) tau.jac(beta)[2,], confun, confun.jac,
    control.outer = list(trace = FALSE))
stopifnot(aout$convergence == 0)
lower <- c(lower, aout$value)
# tau[2] upper
i <- which(taus[,2] == max(taus[,2]))
beta.start <- mout$batch[i, ]
aout <- auglag(beta.start, function(beta) tau(beta)[2],
    function(beta) tau.jac(beta)[2,], confun, confun.jac,
    control.outer = list(trace = FALSE),
    control.optim = list(fnscale = -1))
stopifnot(aout$convergence == 0)
upper <- c(upper, aout$value)

foo <- cbind(lower, upper)
rownames(foo) <- names(delta)
foo
```

```
##                    lower     upper
## (Intercept)   2.21829   5.78171
## x           171.64508 354.76900

tau.hat

## (Intercept)           x
##           4         300
```

A.6.6 For Linear Function of Tau

One last confidence interval, which shows that we really do need to allow arbitrary linear functions of parameters, at least.

```
sum(tau.hat * delta)

## [1] 2.5
```

This is the maximum possible value of $\langle \tau, \delta \rangle$. What is the confidence interval for it?

```
aout <- auglag(beta.start.save,
    function(beta) sum(tau(beta) * delta),
    function(beta) drop(tau.jac(beta) %*% delta),
    confun, confun.jac,
    control.outer = list(trace = FALSE))
stopifnot(aout$convergence == 0)
aout$value

## [1] 1.135503

# just check that this is OK
confun(aout$par)

## [1] -4.864609e-08

sum(tau(aout$par) * delta)

## [1] 1.135503
```

Of course, since the length of a direction of recession is irrelevant (only the direction matters), this is hard to interpret. For that matter, any invertible affine function of the canonical sufficient statistic is another canonical sufficient statistic, so that makes this bound even harder to interpret. So let us get some other values to compare it with.

```
sum(tau.hat * delta)

## [1] 2.5

sum(crossprod(modmat, - y) * delta)

## [1] -2.5

sum(crossprod(modmat, rep(0.5, length(y))) * delta)

## [1] 0
```

So our lower bound is about halfway between the maximum value (the upper endpoint of the confidence interval) and zero, which is the value of this parameter for the model that does not use x and predicts 0.5 for all success probabilities.

B Quasi-Complete Separation Example of Agresti

B.1 Data

Agresti (2013, Section 6.5.1) also discusses another toy problem, also a simple logistic regression.

```
x <- c(x, 50, 50)
y <- c(y, 0, 1)
```

B.2 First Linear Program

```
modmat <- sparse.model.matrix(y ~ x)

tangent.direction <- as.numeric(y == 0) - as.numeric(y == 1)
```

```
objgrd <- rbind(tangent.direction) %*% modmat
objgrd <- as(objgrd, "numeric")

lp <- initProbGLPK()
addRowsGLPK(lp, nrow(modmat))

## [1] 1

addColsGLPK(lp, ncol(modmat))

## [1] 1

setSimplexParmGLPK(MSG_LEV, GLP_MSG_OFF) # STFU
# row bounds
lower.bounds <- pmin(- tangent.direction, 0)
upper.bounds <- pmax(- tangent.direction, 0)
idx <- which(lower.bounds == upper.bounds)
setRowsBndsGLPK(lp, idx, lower.bounds[idx],
    upper.bounds[idx], rep(GLP_FX, length(idx)))
idx <- which(lower.bounds != upper.bounds)
setRowsBndsGLPK(lp, idx, lower.bounds[idx],
    upper.bounds[idx], rep(GLP_DB, length(idx)))
# col bounds: free
idx <- 1:ncol(modmat)
setColsBndsGLPK(lp, idx, rep(-Inf, length(idx)),
    rep(Inf, length(idx)), rep(GLP_FR, length(idx)))
# objective function
setObjDirGLPK(lp, GLP_MIN)
idx <- which(objgrd != 0)
setObjCoefsGLPK(lp, idx, objgrd[idx])
# constraint matrix (modmat)
foompter <- mat2triplet(modmat)
loadMatrixGLPK(lp, nnzero(modmat), foompter$i, foompter$j, foompter$x)

# do it
solveSimplexGLPK(lp)

## [1] 0

# have solution?
getPrimStatGLPK(lp) == GLP_FEAS
```

```
## [1] TRUE

getObjValGLPK(lp)

## [1] -5
```

Because the optimal value is negative, the MLE does not exist in the OM, and the solution is a DOR.

```
delta <- getColsPrimGLPK(lp)
eta <- getRowsPrimGLPK(lp)
names(delta) <- colnames(modmat)
names(eta) <- x

delta

## (Intercept)           x
##      -1.250       0.025

eta

##    10    20    30    40    60    70    80    90    50
## -1.00 -0.75 -0.50 -0.25  0.25  0.50  0.75  1.00  0.00
##    50
##  0.00

all(eta * tangent.direction <= 0)

## [1] TRUE

all(tangent.direction != 0 | eta == 0)

## [1] TRUE
```

B.3 Second Linear Program

Now we have two components of η that we cannot tell whether they are zero or not.

```
eta[abs(eta) < 0.001]

## 50 50
##  0  0
```

Our linear programming software says exactly zero, but maybe that is inaccuracy of computer arithmetic. Thus we are not even sure what a DOR for this problem is, at least not yet.

Hence we do need to do another linear program.

```
is.unknown <- abs(eta) < 0.001
objgrd <- rbind(is.unknown * tangent.direction) %*% modmat
objgrd

## 1 x 2 Matrix of class "dgeMatrix"
##      (Intercept) x
## [1,]           0 0
```

We follow the hint in Section 8.6.2 above that if the objective function is the zero function, then the optimal value can only be zero, so we are done. The LCM only conditions the components of the response vector corresponding to $x \neq 50$ to equal their observed values. The components of the response vector corresponding to $x = 50$ stay random in the LCM.

B.4 Limiting Conditional Model

Fit the LCM.

```
gout <- glm(y ~ x, family = binomial, subset = is.unknown)
cout <- summary(gout)$coef
coefs <- matrix(NA_real_, ncol(modmat), ncol(cout))
coefs[colnames(modmat) %in% rownames(cout), ] <- cout
coefs <- cbind(coefs[ , 1], delta, coefs[ , -1])
rownames(coefs) <- colnames(modmat)
colnames(coefs) <- c("Estimate", "GDOR", "Std. Error", "z value", "Pr(>|z|)")
printCoefmat(coefs)

##                Estimate        GDOR  Std. Error z value Pr(>|z|)
## (Intercept)  4.7103e-16 -1.2500e+00  1.4142e+00       0        1
## x                    NA  2.5000e-02          NA      NA       NA
```

Again, this may not be very helpful to users. More useful is the information that the LCM conditions the following components of the response vector to be equal to their observed values

```
! is.unknown

##    10    20    30    40    60    70    80    90    50
##  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
##    50
## FALSE
```

B.5 Clean Up

```
delProbGLPK(lp)
```

B.6 Proof Level 1

```
eta

##    10    20    30    40    60    70    80    90    50
## -1.00 -0.75 -0.50 -0.25  0.25  0.50  0.75  1.00  0.00
##    50
##  0.00
```

Because we are unsure about which components of **eta** are zero, we do **eta** exactly using the method of Section 8.6.1 above. We suspect that the zero components of **eta** are

```
is.zero <- zapsmall(eta) == 0
is.zero

##    10    20    30    40    60    70    80    90    50
## FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
##    50
##  TRUE
```

Thus the matrix Z in the linear equations (16) is

```
z <- modmat[is.zero, , drop = FALSE]
z <- as(z, "matrix")
z <- gmp::as.bigq(z)
z

## Big Rational ('bigq') 2 x 2 matrix:
##      [,1] [,2]
## [1,] 1    50
## [2,] 1    50
```

So because we have integer-valued covariates, we have an exact Z matrix and this method will work. The γ in (16) which was computed using inexact arithmetic is

```
gamma <- gmp::as.bigq(delta)
gamma

## Big Rational ('bigq') object of length 2:
## [1] -5/4
## [2] 3602879701896397/144115188075855872
```

That this is not "simple" is the whole problem we are working on.

The variables in the linear equations (16) are x and λ. We put x first and then λ in the state vector. Let us rewrite (16) as one equation with partitioned matrices

$$\begin{pmatrix} I & Z^T \\ Z & 0 \end{pmatrix} \begin{pmatrix} x \\ \lambda \end{pmatrix} = \begin{pmatrix} \gamma \\ 0 \end{pmatrix}$$

where I is the identity matrix and 0 is the zero matrix of appropriate dimensions.

Do it.

```
a <- cbind2(diag(length(gamma)), t(z))
a <- rbind2(a, cbind2(z, 0 * diag(nrow(z))))
a

## Big Rational ('bigq') 4 x 4 matrix:
##      [,1] [,2] [,3] [,4]
## [1,] 1    0    1    1
## [2,] 0    1    50   50
## [3,] 1    50   0    0
## [4,] 1    50   0    0

# work around bug (reported) in R package gmp
b <- c(delta, rep(0, nrow(z)))
b <- gmp::as.bigq(b)
b

## Big Rational ('bigq') object of length 4:
## [1] -5/4
## [2] 3602879701896397/144115188075855872
## [3] 0
## [4] 0

x <- solve(a, b)

## Error in solve.bigq(a, b):  System is singular
```

Yes, of course. We have the same linear constraint on x twice. So our Z should have only one row, and there should be only one Lagrange multiplier.

```
z <- z[1, , drop = FALSE]
z

## Big Rational ('bigq') 1 x 2 matrix:
##      [,1] [,2]
## [1,] 1    50

a <- cbind2(diag(length(gamma)), t(z))
a <- rbind2(a, cbind2(z, 0 * diag(nrow(z))))
a

## Big Rational ('bigq') 3 x 3 matrix:
##      [,1] [,2] [,3]
## [1,] 1    0    1
## [2,] 0    1    50
## [3,] 1    50   0

b <- c(delta, rep(0, nrow(z)))
b <- gmp::as.bigq(b)
b
```

```
## Big Rational ('bigq') object of length 3:
## [1] -5/4
## [2] 3602879701896397/144115188075855872
## [3] 0

delta <- solve(a, b)
delta <- delta[seq(along = gamma)]
delta

## Big Rational ('bigq') object of length 2:
## [1] -22527005336107220925/18021604268857767936
## [2] 9010802134442888397/360432085377715535872

modmat <- as(modmat, "matrix")
modmat <- gmp::as.bigq(modmat)
modmat

## Big Rational ('bigq') 10 x 2 matrix:
##       [,1] [,2]
##  [1,] 1    10
##  [2,] 1    20
##  [3,] 1    30
##  [4,] 1    40
##  [5,] 1    60
##  [6,] 1    70
##  [7,] 1    80
##  [8,] 1    90
##  [9,] 1    50
## [10,] 1    50

eta <- gmp::`%*%`(modmat, delta)
eta

## Big Rational ('bigq') 10 x 1 matrix:
##       [,1]
##  [1,] -45054010672214441985/45054010672214441984
##  [2,] -135162032016643325955/18021604268857767936
##  [3,] -45054010672214441985/90108021344428883968
##  [4,] -45054010672214441985/18021604268857767936
##  [5,] 45054010672214441985/18021604268857767936
##  [6,] 4505401067221444 1985/9010802134442888 3968
```

```
##  [6,] 45054010672214441985/90108021344428883968
##  [7,] 135162032016643325955/180216042688857767936
##  [8,] 45054010672214441985/45054010672214441984
##  [9,] 0
## [10,] 0

all(eta * tangent.direction <= 0)

## [1] TRUE

all(tangent.direction != 0 | eta == 0)

## [1] TRUE
```

Since this `eta` has the correct signs, we have proved the `delta` it came from is a DOR. Hence the MLE for the OM does not exist.

We could also get a nicer DOR by just guessing.

```
delta <- gmp::asNumeric(delta)
delta

## [1] -1.250  0.025

delta <- delta / min(abs(delta))
delta

## [1] -50   1

delta <- round(delta)

modmat <- gmp::asNumeric(modmat)
drop(modmat %*% delta)

##  [1] -40 -30 -20 -10  10  20  30  40   0   0
```

But just guessing is not a method guaranteed to work.

B.7 Proof Level 2

Now we want to redo the last linear program in exact rational arithmetic using R function `lpcdd` in R package `rcdd` as suggested in Section 8.6.2 above. From (6) we see the objective function for this program is

```
objgrd <- rbind(is.unknown * tangent.direction) %*% modmat
objgrd

##      [,1] [,2]
## [1,]    0    0
```

We don't have to go any farther. Obviously any linear program that has a solution and has objective function that is the zero function has optimal value zero (if the feasible region is nonempty, and we know it is nonempty for all the linear programs in this document, then any feasible point is a solution).

C Clinical Trial Example of Agresti

C.1 Data

Agresti (2013, Section 6.5.2, Table 6.11) gives the following data.

```
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 <- success
n <- success + failure
```

C.2 First Linear Program

```
modmat <- sparse.model.matrix(~ center + treatment)

tangent.direction <- as.numeric(y == 0) - as.numeric(y == n)

objgrd <- rbind(tangent.direction) %*% modmat
objgrd <- as(objgrd, "numeric")

lp <- initProbGLPK()
addRowsGLPK(lp, nrow(modmat))
```

```
## [1] 1

addColsGLPK(lp, ncol(modmat))

## [1] 1

setSimplexParmGLPK(MSG_LEV, GLP_MSG_OFF) # STFU
# row bounds
lower.bounds <- pmin(- tangent.direction, 0)
upper.bounds <- pmax(- tangent.direction, 0)
idx <- which(lower.bounds == upper.bounds)
setRowsBndsGLPK(lp, idx, lower.bounds[idx],
    upper.bounds[idx], rep(GLP_FX, length(idx)))
idx <- which(lower.bounds != upper.bounds)
setRowsBndsGLPK(lp, idx, lower.bounds[idx],
    upper.bounds[idx], rep(GLP_DB, length(idx)))
# col bounds: free
idx <- 1:ncol(modmat)
setColsBndsGLPK(lp, idx, rep(-Inf, length(idx)),
    rep(Inf, length(idx)), rep(GLP_FR, length(idx)))
# objective function
setObjDirGLPK(lp, GLP_MIN)
idx <- which(objgrd != 0)
setObjCoefsGLPK(lp, idx, objgrd[idx])
# constraint matrix (modmat)
foompter <- mat2triplet(modmat)
loadMatrixGLPK(lp, nnzero(modmat), foompter$i, foompter$j, foompter$x)

# do it
solveSimplexGLPK(lp)

## [1] 0

# have solution?
getPrimStatGLPK(lp) == GLP_FEAS

## [1] TRUE

getObjValGLPK(lp)

## [1] -4
```

Because the optimal value is negative, the MLE does not exist in the OM, and the solution is a DOR.

```
delta <- getColsPrimGLPK(lp)
eta <- getRowsPrimGLPK(lp)
names(delta) <- colnames(modmat)

delta

##      (Intercept)          center2          center3
##               -1                1                0
##          center4          center5 treatmentplacebo
##                1                1                0

eta

##  [1] -1 -1  0  0 -1 -1  0  0  0  0

all(eta * tangent.direction <= 0)

## [1] TRUE

all(tangent.direction != 0 | eta == 0)

## [1] TRUE
```

C.3 Second Linear Program

We are not done because not all components of η that can be nonzero are.

```
eta

##  [1] -1 -1  0  0 -1 -1  0  0  0  0

tangent.direction

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

So we do another linear program.

```
is.unknown <- abs(eta) < 0.001
objgrd <- rbind(is.unknown * tangent.direction) %*% modmat
objgrd <- as(objgrd, "numeric")
idx <- 1:length(objgrd)
setObjCoefsGLPK(lp, idx, objgrd)
lower.bounds[(! is.unknown) & (y == 0)] <- -Inf
upper.bounds[(! is.unknown) & (y == n)] <- Inf
idx <- which(is.infinite(lower.bounds))
setRowsBndsGLPK(lp, idx, lower.bounds[idx],
    upper.bounds[idx], rep(GLP_UP, length(idx)))
idx <- which(is.infinite(upper.bounds))
setRowsBndsGLPK(lp, idx, lower.bounds[idx],
    upper.bounds[idx], rep(GLP_LO, length(idx)))

# do it
solveSimplexGLPK(lp)

## [1] 0

# have solution?
getPrimStatGLPK(lp) == GLP_FEAS

## [1] TRUE

getObjValGLPK(lp)

## [1] 0
```

So, because the optimal value is zero, we are done. The solution we found in the first linear program is a GDOR.

C.4 Limiting Conditional Model

Fit the LCM

```
gout <- glm(cbind(success, failure) ~ center + treatment,
    family = binomial, subset = abs(eta) < 0.001)
cout <- summary(gout)$coef
coefs <- matrix(NA_real_, ncol(modmat), ncol(cout))
coefs[colnames(modmat) %in% rownames(cout), ] <- cout
```

```
coefs <- cbind(coefs[ , 1], zapsmall(delta), coefs[ , -1])
rownames(coefs) <- colnames(modmat)
colnames(coefs) <- c("Estimate", "GDOR", "Std. Error", "z value", "Pr(>|z|)")
printCoefmat(coefs)

##                  Estimate     GDOR Std. Error z value Pr(>|z|)
## (Intercept)      -2.65654 -1.00000    1.03604 -2.5641  0.01034 *
## center2                NA  1.00000         NA      NA       NA
## center3                NA  0.00000         NA      NA       NA
## center4           3.24335  1.00000    1.17197  2.7674  0.00565 **
## center5           2.18022  1.00000    1.13273  1.9247  0.05426 .
## treatmentplacebo -1.54600  0.00000    0.70165 -2.2034  0.02757 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

So this says we don't really have a problem from the MLE not existing in the OM. Despite some centers not contributing information to the conclusion, we still seem to have a (somewhat) statistically significant treatment effect.

If we look at the saturated model statistics and parameters we get

```
beta <- coefs[ , "Estimate"]
beta[is.na(beta)] <- 0

theta <- modmat %*% beta
theta <- as(theta, "numeric")
theta

##  [1] -2.6565436 -4.2025399 -2.6565436 -4.2025399
##  [5] -2.6565436 -4.2025399  0.5868032 -0.9591931
##  [9] -0.4763235 -2.0223197

eta

##  [1] -1 -1  0  0 -1 -1  0  0  0  0

pi <- 1 / (1 + exp(- theta))
pi[eta < 0] <- 0
pi[eta > 0] <- 1
pi
```

```
##  [1] 0.00000000 0.00000000 0.06558684 0.01473711
##  [5] 0.00000000 0.00000000 0.64263131 0.27703978
##  [9] 0.38312066 0.11687934
```

```
print(data.frame(success, failure, treatment, center, theta, eta, pi),
    digits = 2)
```

```
##    success failure   treatment center theta eta    pi
## 1        0       5 active_drug      1 -2.66  -1 0.000
## 2        0       9     placebo      1 -4.20  -1 0.000
## 3        1      12 active_drug      2 -2.66   0 0.066
## 4        0      10     placebo      2 -4.20   0 0.015
## 5        0       7 active_drug      3 -2.66  -1 0.000
## 6        0       5     placebo      3 -4.20  -1 0.000
## 7        6       3 active_drug      4  0.59   0 0.643
## 8        2       6     placebo      4 -0.96   0 0.277
## 9        5       9 active_drug      5 -0.48   0 0.383
## 10       2      12     placebo      5 -2.02   0 0.117
```

Right! The LCM just ignores the data for centers 1 and 3 where there were no successes in either treatment group. If we wanted a confidence interval for the treatment effect that uses the data for all centers, Geyer (2009) has methods for that. Eventually we need to implement them for this package. But the inference made above, which ignores centers 1 and 3 is actually valid (not derived by data snooping, but rather by the notion that there should be center effects).

C.5 Clean Up

```
delProbGLPK(lp)
```

D A Product Multinomial Example

D.1 Data

Not having a handy multinomial example exhibiting directions of recession, we make one up. This is just like the example of Section A except we make the response multinomial rather than binomial.

```
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
```

D.2 Model Matrices and Response Vector

As in Section A we start with the model matrix that goes with the formula `~ x`

```
modmat <- model.matrix(~ x)
modmat

##   (Intercept)  x
## 1           1 10
## 2           1 20
## 3           1 30
## 4           1 40
## 5           1 50
## 6           1 60
## 7           1 70
## 8           1 80
## 9           1 90
## attr(,"assign")
## [1] 0 1
```

But this is not the model matrix we use to make this equivalent to a Poisson sampling model. For that we start with a list

```
foo <- list(modmat, modmat, modmat)
for (i in seq(along = foo)) {
    n <- colnames(modmat)
    n <- paste(n, levels(y)[i], sep = ":y")
    n <- sub("(Intercept):", "", n, fixed = TRUE)
    colnames(foo[[i]]) <- n
}
foo
```

```
## [[1]]
##   yred x:yred
## 1    1     10
## 2    1     20
## 3    1     30
## 4    1     40
## 5    1     50
## 6    1     60
## 7    1     70
## 8    1     80
## 9    1     90
## attr(,"assign")
## [1] 0 1
##
## [[2]]
##   ygreen x:ygreen
## 1      1       10
## 2      1       20
## 3      1       30
## 4      1       40
## 5      1       50
## 6      1       60
## 7      1       70
## 8      1       80
## 9      1       90
## attr(,"assign")
## [1] 0 1
##
## [[3]]
##   yblue x:yblue
## 1     1      10
## 2     1      20
## 3     1      30
## 4     1      40
## 5     1      50
## 6     1      60
## 7     1      70
## 8     1      80
## 9     1      90
## attr(,"assign")
## [1] 0 1
```

```
modmat <- bdiag(foo)
colnames(modmat) <- sapply(foo, colnames)

modmat
```

```
## 27 x 6 sparse Matrix of class "dgCMatrix"
##        yred x:yred ygreen x:ygreen yblue x:yblue
##  [1,]    1     10      .        .     .       .
##  [2,]    1     20      .        .     .       .
##  [3,]    1     30      .        .     .       .
##  [4,]    1     40      .        .     .       .
##  [5,]    1     50      .        .     .       .
##  [6,]    1     60      .        .     .       .
##  [7,]    1     70      .        .     .       .
##  [8,]    1     80      .        .     .       .
##  [9,]    1     90      .        .     .       .
## [10,]    .      .      1       10     .       .
## [11,]    .      .      1       20     .       .
## [12,]    .      .      1       30     .       .
## [13,]    .      .      1       40     .       .
## [14,]    .      .      1       50     .       .
## [15,]    .      .      1       60     .       .
## [16,]    .      .      1       70     .       .
## [17,]    .      .      1       80     .       .
## [18,]    .      .      1       90     .       .
## [19,]    .      .      .        .     1      10
## [20,]    .      .      .        .     1      20
## [21,]    .      .      .        .     1      30
## [22,]    .      .      .        .     1      40
## [23,]    .      .      .        .     1      50
## [24,]    .      .      .        .     1      60
## [25,]    .      .      .        .     1      70
## [26,]    .      .      .        .     1      80
## [27,]    .      .      .        .     1      90
```

Now we have to turn factor response into matrix response and then vector response.

```
y <- model.matrix(~ 0 + y)
y

##   yred ygreen yblue
## 1    1      0     0
## 2    1      0     0
## 3    1      0     0
## 4    0      1     0
## 5    0      1     0
## 6    0      1     0
## 7    0      0     1
## 8    0      0     1
## 9    0      0     1
## attr(,"assign")
## [1] 1 1 1
## attr(,"contrasts")
## attr(,"contrasts")$y
## [1] "contr.treatment"
```

This is the response matrix. The strata are the row names which we have to make up.

```
strata <- paste0("row", row(y))
strata

##  [1] "row1" "row2" "row3" "row4" "row5" "row6" "row7"
##  [8] "row8" "row9" "row1" "row2" "row3" "row4" "row5"
## [15] "row6" "row7" "row8" "row9" "row1" "row2" "row3"
## [22] "row4" "row5" "row6" "row7" "row8" "row9"
```

Now we turn the response matrix into a vector and the strata into a model matrix.

```
y.matrix <- as.matrix(y) # save for future reference
y <- as.vector(y)
modmat.strata <- sparse.model.matrix(~ 0 + strata)
colnames(modmat.strata) <- sub("strata", "", colnames(modmat.strata))
modmat.strata

## 27 x 9 sparse Matrix of class "dgCMatrix"
##    row1 row2 row3 row4 row5 row6 row7 row8 row9
## 1     1    .    .    .    .    .    .    .    .
## 2     .    1    .    .    .    .    .    .    .
## 3     .    .    1    .    .    .    .    .    .
## 4     .    .    .    1    .    .    .    .    .
## 5     .    .    .    .    1    .    .    .    .
## 6     .    .    .    .    .    1    .    .    .
## 7     .    .    .    .    .    .    1    .    .
## 8     .    .    .    .    .    .    .    1    .
## 9     .    .    .    .    .    .    .    .    1
## 10    1    .    .    .    .    .    .    .    .
## 11    .    1    .    .    .    .    .    .    .
## 12    .    .    1    .    .    .    .    .    .
## 13    .    .    .    1    .    .    .    .    .
## 14    .    .    .    .    1    .    .    .    .
## 15    .    .    .    .    .    1    .    .    .
## 16    .    .    .    .    .    .    1    .    .
## 17    .    .    .    .    .    .    .    1    .
## 18    .    .    .    .    .    .    .    .    1
## 19    1    .    .    .    .    .    .    .    .
## 20    .    1    .    .    .    .    .    .    .
## 21    .    .    1    .    .    .    .    .    .
## 22    .    .    .    1    .    .    .    .    .
## 23    .    .    .    .    1    .    .    .    .
## 24    .    .    .    .    .    1    .    .    .
## 25    .    .    .    .    .    .    1    .    .
## 26    .    .    .    .    .    .    .    1    .
## 27    .    .    .    .    .    .    .    .    1
```

Now we put this together with the other model matrix and run Algorithm 1.

```
modmat.covariates <- modmat
modmat <- cbind2(modmat.strata, modmat.covariates)
modmat

## 27 x 15 sparse Matrix of class "dgCMatrix"

##   [[ suppressing 15 column names 'row1', 'row2', 'row3' ...  ]]

##
## 1  1 . . . . . . . . 1 10 .  . .  .
## 2  . 1 . . . . . . . 1 20 .  . .  .
## 3  . . 1 . . . . . . 1 30 .  . .  .
## 4  . . . 1 . . . . . 1 40 .  . .  .
## 5  . . . . 1 . . . . 1 50 .  . .  .
## 6  . . . . . 1 . . . 1 60 .  . .  .
## 7  . . . . . . 1 . . 1 70 .  . .  .
## 8  . . . . . . . 1 . 1 80 .  . .  .
## 9  . . . . . . . . 1 1 90 .  . .  .
## 10 1 . . . . . . . . .  . 1 10 .  .
## 11 . 1 . . . . . . . .  . 1 20 .  .
## 12 . . 1 . . . . . . .  . 1 30 .  .
## 13 . . . 1 . . . . . .  . 1 40 .  .
## 14 . . . . 1 . . . . .  . 1 50 .  .
## 15 . . . . . 1 . . . .  . 1 60 .  .
## 16 . . . . . . 1 . . .  . 1 70 .  .
## 17 . . . . . . . 1 . .  . 1 80 .  .
## 18 . . . . . . . . 1 .  . 1 90 .  .
## 19 1 . . . . . . . . .  . .  . 1 10
## 20 . 1 . . . . . . . .  . .  . 1 20
## 21 . . 1 . . . . . . .  . .  . 1 30
## 22 . . . 1 . . . . . .  . .  . 1 40
## 23 . . . . 1 . . . . .  . .  . 1 50
## 24 . . . . . 1 . . . .  . .  . 1 60
## 25 . . . . . . 1 . . .  . .  . 1 70
## 26 . . . . . . . 1 . .  . .  . 1 80
## 27 . . . . . . . . 1 .  . .  . 1 90
```

D.3 First Linear Program

Here we start being more careful to follow Algorithm 1 exactly. (Of course, we were following Algorithm 2 before, but we were not using all of the same terminology.)

```
tangent.direction <- as.numeric(y == 0)

i.double.star <- rep(FALSE, length(y))
i.triple.star <- y == 0
gamma <- rep(0, ncol(modmat))

objgrd <- rbind(i.triple.star * tangent.direction) %*% modmat
objgrd <- as(objgrd, "numeric")

lp <- initProbGLPK()
addRowsGLPK(lp, nrow(modmat))

## [1] 1

addColsGLPK(lp, ncol(modmat))

## [1] 1

setSimplexParmGLPK(MSG_LEV, GLP_MSG_OFF) # STFU
# row bounds
lower.bounds <- pmin(- tangent.direction, 0)
upper.bounds <- pmax(- tangent.direction, 0)
idx <- which(lower.bounds == upper.bounds)
setRowsBndsGLPK(lp, idx, lower.bounds[idx],
    upper.bounds[idx], rep(GLP_FX, length(idx)))
idx <- which(lower.bounds != upper.bounds)
setRowsBndsGLPK(lp, idx, lower.bounds[idx],
    upper.bounds[idx], rep(GLP_DB, length(idx)))
# col bounds: free
idx <- 1:ncol(modmat)
setColsBndsGLPK(lp, idx, rep(-Inf, length(idx)),
    rep(Inf, length(idx)), rep(GLP_FR, length(idx)))
# objective function
setObjDirGLPK(lp, GLP_MIN)
idx <- which(objgrd != 0)
```

```
setObjCoefsGLPK(lp, idx, objgrd[idx])
# constraint matrix (modmat)
foompter <- mat2triplet(modmat)
loadMatrixGLPK(lp, nnzero(modmat), foompter$i, foompter$j, foompter$x)

# do it
solveSimplexGLPK(lp)

## [1] 0

# have solution?
getPrimStatGLPK(lp) == GLP_FEAS

## [1] TRUE

getObjValGLPK(lp)

## [1] -6.75
```

Because the optimal value is negative, the MLE does not exist in the OM, and the solution is a DOR.

```
delta <- getColsPrimGLPK(lp)
eta <- getRowsPrimGLPK(lp)
names(delta) <- colnames(modmat)

gamma <- delta + gamma

delta

##          row1          row2          row3
## -1.250000e-01  0.000000e+00  1.250000e-01
##          row4          row5          row6
##  1.250000e-01  1.250000e-01  1.250000e-01
##          row7          row8          row9
##  1.250000e-01  0.000000e+00 -1.250000e-01
##          yred        x:yred        ygreen
##  2.500000e-01 -1.250000e-02 -1.250000e-01
##      x:ygreen         yblue       x:yblue
##  4.163336e-18 -1.000000e+00  1.250000e-02
```

```
eta

##  [1]  0.000  0.000  0.000 -0.125 -0.250 -0.375 -0.500
##  [8] -0.750 -1.000 -0.250 -0.125  0.000  0.000  0.000
## [15]  0.000  0.000 -0.125 -0.250 -1.000 -0.750 -0.500
## [22] -0.375 -0.250 -0.125  0.000  0.000  0.000

max(eta)

## [1] 0

i.double.star <- i.double.star | abs(eta) > 0.001
i.triple.star <- i.triple.star & (! i.double.star)

i.double.star

##  [1] FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [9]  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE
## [17]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [25] FALSE FALSE FALSE

i.triple.star

##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [9] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE
## [17] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [25] FALSE FALSE FALSE
```

D.4 Second Linear Program

We are not done because the optimal value of the last linear program done was not zero and I^{***} is not the empty set. So we do another linear program.

```
objgrd <- rbind(i.triple.star * tangent.direction) %*% modmat
objgrd <- as(objgrd, "numeric")
idx <- 1:length(objgrd)
setObjCoefsGLPK(lp, idx, objgrd)
lower.bounds[(! i.triple.star) & (y == 0)] <- -Inf
idx <- which(is.infinite(lower.bounds))
```

```
setRowsBndsGLPK(lp, idx, lower.bounds[idx],
    upper.bounds[idx], rep(GLP_UP, length(idx)))

# do it
solveSimplexGLPK(lp)

## [1] 0

# have solution?
getPrimStatGLPK(lp) == GLP_FEAS

## [1] TRUE

getObjValGLPK(lp)

## [1] -2
```

Because the optimal value is negative we aren't done yet.

```
delta <- getColsPrimGLPK(lp)
eta <- getRowsPrimGLPK(lp)
names(delta) <- colnames(modmat)

gamma <- delta + gamma

delta

##          row1          row2          row3
## -1.000000e+00  0.000000e+00  1.000000e+00
##          row4          row5          row6
##  2.000000e+00  2.000000e+00  2.000000e+00
##          row7          row8          row9
##  1.000000e+00  0.000000e+00 -1.000000e+00
##          yred        x:yred        ygreen
##  2.000000e+00 -1.000000e-01 -2.000000e+00
##      x:ygreen         yblue       x:yblue
## -1.850372e-17 -8.000000e+00  1.000000e-01

eta

##  [1]  0  0  0  0 -1 -2 -4 -6 -8 -3 -2 -1  0  0  0 -1 -2
## [18] -3 -8 -6 -4 -2 -1  0  0  0  0

max(eta)

## [1] 0

i.double.star <- i.double.star | abs(eta) > 0.001
i.triple.star <- i.triple.star & (! i.double.star)

i.double.star

##  [1] FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [9]  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE
## [17]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [25] FALSE FALSE FALSE

i.triple.star

##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [9] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [17] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [25] FALSE FALSE FALSE

all(! i.triple.star)

## [1] TRUE
```

Now we quit because I^{***} is empty, so γ is a GDOR and I^{**} is the index set of the components of the response vector that are conditioned to be equal to their observed values in the LCM.

D.5 Limiting Conditional Model

Because of the multinomial response, it is a bit confusing what we have found. Let us put I^{**} in matrix form (to compare with the "response matrix").

```
matrix(i.double.star, nrow = nrow(y.matrix), dimnames = dimnames(y.matrix))

##    yred ygreen yblue
## 1 FALSE   TRUE  TRUE
## 2 FALSE   TRUE  TRUE
## 3 FALSE   TRUE  TRUE
## 4  TRUE  FALSE  TRUE
## 5  TRUE  FALSE  TRUE
## 6  TRUE  FALSE  TRUE
## 7  TRUE   TRUE FALSE
## 8  TRUE   TRUE FALSE
## 9  TRUE   TRUE FALSE
```

So we see that, despite η not having all components nonzero, the LCM is completely degenerate. As Theorem 15 says, if the matrix above has all but one component in a row `TRUE` (meaning the corresponding component of the response is conditioned in the LCM to be zero), then the other component in the row must be conditioned to be equal to its value too (which is n_A for row A, also called stratum A).

We do report the summary

```
coefs <- cbind(NA, gamma, NA, NA, NA)
rownames(coefs) <- colnames(modmat)
colnames(coefs) <- c("Estimate", "GDOR", "Std. Error", "z value", "Pr(>|z|)")
printCoefmat(coefs)

##           Estimate       GDOR Std. Error z value
## row1            NA -1.125e+00         NA      NA
## row2            NA  0.000e+00         NA      NA
## row3            NA  1.125e+00         NA      NA
## row4            NA  2.125e+00         NA      NA
## row5            NA  2.125e+00         NA      NA
## row6            NA  2.125e+00         NA      NA
## row7            NA  1.125e+00         NA      NA
## row8            NA  0.000e+00         NA      NA
## row9            NA -1.125e+00         NA      NA
## yred            NA  2.250e+00         NA      NA
## x:yred          NA -1.125e-01         NA      NA
## ygreen          NA -2.125e+00         NA      NA
## x:ygreen        NA -1.434e-17         NA      NA
## yblue           NA -9.000e+00         NA      NA
## x:yblue         NA  1.125e-01         NA      NA
##           Pr(>|z|)
## row1            NA
## row2           NA
## row3           NA
## row4           NA
## row5           NA
## row6           NA
## row7           NA
## row8           NA
## row9           NA
## yred           NA
## x:yred         NA
## ygreen         NA
## x:ygreen       NA
## yblue          NA
## x:yblue        NA
```

Except this is wrong. Columns of `modmat.strata` are DOC of the OM and we should not report either their coefficients (if the LCM weren't completely degenerate) or the corresponding components of the GDOR.

```
coefs <- coefs[- seq(1, ncol(modmat.strata)), ]
printCoefmat(zapsmall(coefs))

##            Estimate    GDOR Std. Error z value Pr(>|z|)
## yred             NA  2.2500         NA      NA       NA
## x:yred           NA -0.1125         NA      NA       NA
## ygreen           NA -2.1250         NA      NA       NA
## x:ygreen         NA  0.0000         NA      NA       NA
## yblue            NA -9.0000         NA      NA       NA
## x:yblue          NA  0.1125         NA      NA       NA
```

D.6 Clean Up

```
delProbGLPK(lp)
```

E Another Product Multinomial Example

E.1 Data

Having done one product multinomial example with a completely degenerate LCM, we do another with an only partially degenerate LCM. We use the same `x` and the same formula as in the preceding example. We just change y.

```
y <- rep(c("red", "green", "blue"), each = 3)
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
```

E.2 Model Matrices and Response Vector

As we said above, we do not need to redo the model matrices. But we have to turn factor response into matrix response and then vector response.

```
y <- model.matrix(~ 0 + y)
y.matrix <- as.matrix(y) # save for future reference
y <- as.vector(y)
```

This is the response matrix. The strata are unchanged from the preceding section.

E.3 First Linear Program

Very similar to the preceding example.

```
tangent.direction <- as.numeric(y == 0)

i.double.star <- rep(FALSE, length(y))
i.triple.star <- y == 0
gamma <- rep(0, ncol(modmat))

objgrd <- rbind(i.triple.star * tangent.direction) %*% modmat
objgrd <- as(objgrd, "numeric")

lp <- initProbGLPK()
addRowsGLPK(lp, nrow(modmat))

## [1] 1

addColsGLPK(lp, ncol(modmat))

## [1] 1

setSimplexParmGLPK(MSG_LEV, GLP_MSG_OFF) # STFU
# row bounds
lower.bounds <- pmin(- tangent.direction, 0)
upper.bounds <- pmax(- tangent.direction, 0)
idx <- which(lower.bounds == upper.bounds)
setRowsBndsGLPK(lp, idx, lower.bounds[idx],
    upper.bounds[idx], rep(GLP_FX, length(idx)))
idx <- which(lower.bounds != upper.bounds)
setRowsBndsGLPK(lp, idx, lower.bounds[idx],
    upper.bounds[idx], rep(GLP_DB, length(idx)))
# col bounds: free
idx <- 1:ncol(modmat)
setColsBndsGLPK(lp, idx, rep(-Inf, length(idx)),
    rep(Inf, length(idx)), rep(GLP_FR, length(idx)))
# objective function
setObjDirGLPK(lp, GLP_MIN)
idx <- which(objgrd != 0)
setObjCoefsGLPK(lp, idx, objgrd[idx])
# constraint matrix (modmat)
```

```
foompter <- mat2triplet(modmat)
loadMatrixGLPK(lp, nnzero(modmat), foompter$i, foompter$j, foompter$x)

# do it
solveSimplexGLPK(lp)

## [1] 0

# have solution?
getPrimStatGLPK(lp) == GLP_FEAS

## [1] TRUE

getObjValGLPK(lp)

## [1] -5.4
```

Because the optimal value is negative, the MLE does not exist in the OM, and the solution is a DOR.

```
delta <- getColsPrimGLPK(lp)
eta <- getRowsPrimGLPK(lp)
names(delta) <- colnames(modmat)

gamma <- delta + gamma

delta

##         row1         row2         row3         row4
## -0.133333333  0.000000000  0.133333333  0.266666667
##         row5         row6         row7         row8
##  0.200000000  0.133333333  0.066666667  0.000000000
##         row9         yred       x:yred       ygreen
## -0.066666667  0.266666667 -0.013333333 -0.533333333
##     x:ygreen        yblue      x:yblue
##  0.006666667 -0.533333333  0.006666667

eta

##  [1]  0.000000e+00  0.000000e+00  0.000000e+00
##  [4]  0.000000e+00 -2.000000e-01 -4.000000e-01
##  [7] -6.000000e-01 -8.000000e-01 -1.000000e+00
## [10] -6.000000e-01 -4.000000e-01 -2.000000e-01
## [13]  0.000000e+00  0.000000e+00  0.000000e+00
## [16]  0.000000e+00  0.000000e+00  5.859510e-17
## [19] -6.000000e-01 -4.000000e-01 -2.000000e-01
## [22]  0.000000e+00 -5.551115e-17  0.000000e+00
## [25]  0.000000e+00  0.000000e+00  0.000000e+00

max(eta)

## [1] 5.85951e-17

i.double.star <- i.double.star | abs(eta) > 0.001
i.triple.star <- i.triple.star & (! i.double.star)

i.double.star

##  [1] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
##  [9]  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE
## [17] FALSE FALSE  TRUE  TRUE  TRUE FALSE FALSE FALSE
## [25] FALSE FALSE FALSE

i.triple.star

##  [1] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
##  [9] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
## [17]  TRUE  TRUE FALSE FALSE FALSE  TRUE  TRUE FALSE
## [25]  TRUE FALSE FALSE
```

E.4 Second Linear Program

We are not done because the optimal value of the last linear program done was not zero and I^{***} is not the empty set. So we do another linear program.

```
objgrd <- rbind(i.triple.star * tangent.direction) %*% modmat
objgrd <- as(objgrd, "numeric")
idx <- 1:length(objgrd)
```

```
setObjCoefsGLPK(lp, idx, objgrd)
lower.bounds[(! i.triple.star) & (y == 0)] <- -Inf

idx <- which(is.infinite(lower.bounds))
setRowsBndsGLPK(lp, idx, lower.bounds[idx],
    upper.bounds[idx], rep(GLP_UP, length(idx)))

# do it
solveSimplexGLPK(lp)

## [1] 0

# have solution?
getPrimStatGLPK(lp) == GLP_FEAS

## [1] TRUE

getObjValGLPK(lp)

## [1] -1
```

Because the optimal value is negative we aren't done yet.

```
delta <- getColsPrimGLPK(lp)
eta <- getRowsPrimGLPK(lp)
names(delta) <- colnames(modmat)

gamma <- delta + gamma

delta

##        row1        row2        row3        row4
## -0.83333333  0.00000000  0.83333333  0.66666667
##        row5        row6        row7        row8
##  0.50000000  0.33333333  0.16666667  0.00000000
##        row9        yred      x:yred      ygreen
## -0.16666667  1.66666667 -0.08333333 -1.33333333
##    x:ygreen       yblue     x:yblue
##  0.01666667 -1.33333333  0.01666667

eta
```

```
##  [1]  0.000000e+00  0.000000e+00  0.000000e+00
##  [4] -1.000000e+00 -2.000000e+00 -3.000000e+00
##  [7] -4.000000e+00 -5.000000e+00 -6.000000e+00
## [10] -2.000000e+00 -1.000000e+00  2.664535e-16
## [13]  0.000000e+00  0.000000e+00  0.000000e+00
## [16]  0.000000e+00  0.000000e+00  0.000000e+00
## [19] -2.000000e+00 -1.000000e+00  0.000000e+00
## [22] -1.110223e-16 -1.110223e-16  0.000000e+00
## [25]  0.000000e+00  0.000000e+00  0.000000e+00

max(eta)

## [1] 2.664535e-16

i.double.star <- i.double.star | eta < -0.001
i.triple.star <- i.triple.star & (! i.double.star)

i.double.star

##  [1] FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [9]  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE
## [17] FALSE FALSE  TRUE  TRUE  TRUE FALSE FALSE FALSE
## [25] FALSE FALSE FALSE

i.triple.star

##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [9] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
## [17]  TRUE  TRUE FALSE FALSE FALSE  TRUE  TRUE FALSE
## [25]  TRUE FALSE FALSE
```

E.5 Third Linear Program

We are not done because the optimal value of the last linear program done was not zero and I^{***} is not the empty set. So we do another linear program.

```
objgrd <- rbind(i.triple.star * tangent.direction) %*% modmat
objgrd <- as(objgrd, "numeric")
```

```
idx <- 1:length(objgrd)
setObjCoefsGLPK(lp, idx, objgrd)
lower.bounds[(! i.triple.star) & (y == 0)] <- -Inf

idx <- which(is.infinite(lower.bounds))
setRowsBndsGLPK(lp, idx, lower.bounds[idx],
    upper.bounds[idx], rep(GLP_UP, length(idx)))

# do it
solveSimplexGLPK(lp)

## [1] 0

# have solution?
getPrimStatGLPK(lp) == GLP_FEAS

## [1] TRUE

getObjValGLPK(lp)

## [1] 0
```

And that does it. Optimal value zero means done!

E.6 Clean Up

```
delProbGLPK(lp)
```

E.7 Proof Level 1

```
eta <- modmat %*% gamma
eta <- as(eta, "numeric")

eta

##  [1]  0.000000e+00  2.220446e-16  4.440892e-16
##  [4] -1.000000e+00 -2.200000e+00 -3.400000e+00
##  [7] -4.600000e+00 -5.800000e+00 -7.000000e+00
## [10] -2.600000e+00 -1.400000e+00 -2.000000e-01
## [13]  1.110223e-16  0.000000e+00  4.440892e-16
## [16]  2.220446e-16  2.220446e-16  0.000000e+00
## [19] -2.600000e+00 -1.400000e+00 -2.000000e-01
## [22]  1.110223e-16  0.000000e+00  4.440892e-16
## [25]  2.220446e-16  2.220446e-16  0.000000e+00
```

Because we are unsure about which components of `eta` are zero, we do `eta` exactly using the methods of Section 8.5 above. We suspect that the zero components of `eta` are

```
is.zero <- zapsmall(eta) == 0
is.zero

##  [1]  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE
##  [9] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
## [17]  TRUE  TRUE FALSE FALSE FALSE  TRUE  TRUE  TRUE
## [25]  TRUE  TRUE  TRUE

range(eta[is.zero])

## [1] 0.000000e+00 4.440892e-16

range(eta[! is.zero])

## [1] -7.0 -0.2
```

Looks like we have pretty clear separation, and we might hope that is convincing enough. But hope isn't proof. So on with the proof.

Thus the matrix Z in the linear equations (16) is

```
z <- modmat[is.zero, , drop = FALSE]
z

## 15 x 15 sparse Matrix of class "dgCMatrix"

##   [[ suppressing 15 column names 'row1', 'row2', 'row3' ...  ]]

##
## 1  1 . . . . . . . . 1 10 .  . . .
## 2  . 1 . . . . . . . 1 20 .  . . .
## 3  . . 1 . . . . . . 1 30 .  . .  .
## 13 . . . 1 . . . . . .  . 1 40 .  .
## 14 . . . . 1 . . . . .  . 1 50 .  .
## 15 . . . . . 1 . . . .  . 1 60 .  .
## 16 . . . . . . 1 . . .  . 1 70 .  .
## 17 . . . . . . . 1 . .  . 1 80 .  .
## 18 . . . . . . . . 1 .  . 1 90 .  .
## 22 . . . 1 . . . . . .  . .  . 1 40
## 23 . . . . 1 . . . . .  . .  . 1 50
## 24 . . . . . 1 . . . .  . .  . 1 60
## 25 . . . . . . 1 . . .  . .  . 1 70
## 26 . . . . . . . 1 . .  . .  . 1 80
## 27 . . . . . . . . 1 .  . .  . 1 90
```

We found doing this kind of proof before (Section B.6 above), that Z had redundant rows. There it was trivial to see. There were two rows and they were the same.

To see whether we have the same issue here (and elsewhere, as we shall see when we come to fitting the LCM) we use R function `redundant` in R package `rcdd`.

```
vrep <- makeV(lines = as.matrix(z))
rout <- redundant(vrep)
rout$new.position

##  [1]  1  2  3  4  5  6  7  8  9 10 11  0  0  0  0

z <- z[rout$new.position > 0, , drop = FALSE]
z

## 11 x 15 sparse Matrix of class "dgCMatrix"

##   [[ suppressing 15 column names 'row1', 'row2', 'row3' ...  ]]

##
## 1  1 . . . . . . . . 1 10 .  . .  .
## 2  . 1 . . . . . . . 1 20 .  . .  .
## 3  . . 1 . . . . . . 1 30 .  . .  .
## 13 . . . 1 . . . . . .  . 1 40 .  .
## 14 . . . . 1 . . . . .  . 1 50 .  .
## 15 . . . . . 1 . . . .  . 1 60 .  .
## 16 . . . . . . 1 . . .  . 1 70 .  .
## 17 . . . . . . . 1 . .  . 1 80 .  .
## 18 . . . . . . . . 1 .  . 1 90 .  .
## 22 . . . 1 . . . . . .  . .  . 1 40
## 23 . . . . 1 . . . . .  . .  . 1 50
```

So the last four rows of Z are redundant, and we got rid of them.

```
z <- as.matrix(z)
z <- gmp::as.bigq(z)
a <- cbind2(diag(ncol(z)), t(z))
a <- rbind2(a, cbind2(z, 0 * diag(nrow(z))))
dim(a)
```

```
## [1] 26 26
```

```
b <- c(gamma, rep(0, nrow(z)))
b <- gmp::as.bigq(b)
length(b)
```

```
## [1] 26
```

```
sout <- solve(a, b)
gamma <- sout[seq(along = gamma)]
gamma
```

```
## Big Rational ('bigq') object of length 15:
##  [1] -3489749279256849245З/36100854613001895936
##  [2] 797/36100854613001895936
##  [3] 11632497597522831349/12033618204333965312
##  [4] 108809849301032675137/116581981393963607654
##  [5] 81607386975774505557/116581981393963607654
##  [6] 18134974883505445325/38860660464654535884
##  [7] 27202462325258166397/116581981393963607654
##  [8] -31829/116581981393963607654
##  [9] -9067487441752724254З/38860660464654535884
## [10] 23264995195045661901/12033618204333965312
## [11] -3489749279256849325/36100854613001895936
## [12] -7253989953402178448/38860660464654535884
## [13] 6800615581314542395/29145495348490901913
## [14] -7253989953402178448/38860660464654535884
## [15] 6800615581314542395/29145495348490901913
```

```
modmat.gmp <- as(modmat, "matrix")
modmat.gmp <- gmp::as.bigq(modmat.gmp)
eta <- gmp::`%*%`(modmat.gmp, gamma)
all(eta * tangent.direction <= 0)

## [1] TRUE

all(tangent.direction != 0 | eta == 0)

## [1] TRUE
```

Since this `eta` has the correct signs, we have proved the `gamma` it came from is a DOR. Hence the MLE for the OM does not exist.

E.8 Proof Level 2

Now we want to redo the last linear program in exact rational arithmetic using R function `lpcdd` in R package `rcdd` as suggested in Section 8.5 above. From (6) we see the objective function for this program is

```
objgrd <- colSums(modmat[y == 0 & is.zero, , drop = FALSE])
objgrd

##      row1      row2      row3      row4      row5      row6
##         0         0         0         1         1         1
##      row7      row8      row9      yred    x:yred    ygreen
##         1         1         1         0         0         3
## x:ygreen     yblue   x:yblue
##       230         3       160
```

Unlike what happened in Section B.7 above, we cannot skip solving the linear program because `objgrd` is the zero vector.

R function `lpcdd` in R package `rcdd` only wants constraints of the form $A_1 x \leq b_1$ or $A_2 x = b_2$. So we have to put the constraints of linear programming problem (5) in that form.

$$\begin{aligned} \sum_{j \in J} m_{ij} \delta_j &= 0, \qquad && y_i > 0 \\ \sum_{j \in J} m_{ij} \delta_j &\le 0, \qquad && y_i = 0 \\ - \sum_{j \in J} m_{ij} \delta_j &\le 1, \qquad && y_i = 0 \text{ and } \eta = 0 \end{aligned}$$

(the $\eta = 0$ in the last line refers to the exact η computed in the level one proof).

```
modmat.rcdd <- as(modmat, "matrix")
modmat.rcdd <- d2q(modmat.rcdd)
foompter <- modmat.rcdd[y == 0, , drop = FALSE]
hrep <- makeH(foompter, rep(0, nrow(foompter)))
foompter <- modmat.rcdd[y == 0 & eta == 0, , drop = FALSE]
hrep <- addHin(qneg(foompter), rep(1, nrow(foompter)), hrep)
foompter <- modmat.rcdd[y != 0, , drop = FALSE]
hrep <- addHeq(foompter, rep(0, nrow(foompter)), hrep)
```

```
lout <- lpcdd(hrep, d2q(objgrd))
lout$solution.type

## [1] "Optimal"

lout$optimal.value

## [1] "0"
```

So that proves (a valid mathematical proof) that the DOR found in the preceding section is a GDOR. Thus we use it henceforth.

```
gamma <- as(gamma, "numeric")
eta <- as(eta, "numeric")
gamma

##  [1] -9.666667e-01  2.207704e-17  9.666667e-01
##  [4]  9.333333e-01  7.000000e-01  4.666667e-01
##  [7]  2.333333e-01 -2.730182e-17 -2.333333e-01
## [10]  1.933333e+00 -9.666667e-02 -1.866667e+00
## [13]  2.333333e-02 -1.866667e+00  2.333333e-02
```

```
eta
```

```
##  [1]  0.0  0.0  0.0 -1.0 -2.2 -3.4 -4.6 -5.8 -7.0 -2.6
## [11] -1.4 -0.2  0.0  0.0  0.0  0.0  0.0  0.0 -2.6 -1.4
## [21] -0.2  0.0  0.0  0.0  0.0  0.0  0.0
```

E.9 Limiting Conditional Model

Because of the multinomial response, it is a bit confusing what we have found. Let us put I^{**} in matrix form (to compare with the "response matrix").

```
i.double.star <- eta < 0
```

```
matrix(i.double.star, nrow = nrow(y.matrix), dimnames = dimnames(y.matrix))
```

```
##    yred ygreen yblue
## 1 FALSE   TRUE  TRUE
## 2 FALSE   TRUE  TRUE
## 3 FALSE   TRUE  TRUE
## 4  TRUE  FALSE FALSE
## 5  TRUE  FALSE FALSE
## 6  TRUE  FALSE FALSE
## 7  TRUE  FALSE FALSE
## 8  TRUE  FALSE FALSE
## 9  TRUE  FALSE FALSE
```

As Theorem 15 says, if the matrix above has all but one component in a row `TRUE` (meaning the corresponding component of the response is conditioned in the LCM to be zero), then the other component in the row must be conditioned to be equal to its value too (which is n_A for row A, also called stratum A).

Thus the first three rows say the multinomial distribution for that row is completely degenerate: blue and green cannot occur and red must occur.

But the rest of the rows say the multinomial distribution for that row is only partially degenerate: red cannot occur but either blue or green is possible.

Thus we see that the LCM is not completely degenerate, only partially degenerate.

It may seem like a lot of work to have gotten here when this result is intuitively obvious (at least in hindsight). But in non-toy problems the answer is far from intuitive, hence all the linear programming.

First we need to patch up the result above as described in Theorem 15.

```
is.free <- matrix(! i.double.star, nrow = nrow(y.matrix),
    dimnames = dimnames(y.matrix))
is.free <- apply(is.free, 1, function(x) if (sum(x) == 1) FALSE & x else x)
is.free <- t(is.free)
is.free

##    yred ygreen yblue
## 1 FALSE  FALSE FALSE
## 2 FALSE  FALSE FALSE
## 3 FALSE  FALSE FALSE
## 4 FALSE   TRUE  TRUE
## 5 FALSE   TRUE  TRUE
## 6 FALSE   TRUE  TRUE
## 7 FALSE   TRUE  TRUE
## 8 FALSE   TRUE  TRUE
## 9 FALSE   TRUE  TRUE

is.free <- as.vector(is.free)
```

Now `is.free` says which components of the response vector are "free" (not conditioned to be equal to their observed values) in the LCM.

So we fit the LCM

```
modmat <- as.matrix(modmat)
modmat.lcm <- modmat[is.free, , drop = FALSE]
y.lcm <- y[is.free]
gout <- glm.fit(modmat.lcm, y.lcm, family = poisson())
coef(gout)

##         row1        row2        row3        row4
##          NA          NA          NA -3.08202063
##        row5        row6        row7        row8
## -1.97106106 -1.04202207 -0.43500828 -0.15001968
##        row9        yred      x:yred      ygreen
## -0.04695166          NA          NA  7.89117931
##    x:ygreen       yblue     x:yblue
## -0.12140276          NA          NA
```

But we are not close to done because we finally have to deal with the fact that our model is not Poisson but rather product multinomial. This means two things.

- The regression coefficients with ‘row‘ in their names are not identifiable for the product multinomial model. So we henceforth just ignore them.
- The standard errors we compute cannot come from R function `glm.fit` because we lied to it and told it the model was `poisson()`. Thus we have to compute standard errors by hand.

```
beta.lcm <- coef(gout)[- seq(1, ncol(modmat.strata))]
beta.lcm

##       yred     x:yred     ygreen   x:ygreen      yblue
##         NA         NA  7.8911793 -0.1214028         NA
##    x:yblue
##         NA

gdor.lcm <- gamma[- seq(1, ncol(modmat.strata))]
gdor.lcm

## [1]  1.93333333 -0.09666667 -1.86666667  0.02333333
## [5] -1.86666667  0.02333333

mu <- y
mu[is.free] <- fitted(gout)
matrix(mu, nrow = nrow(y.matrix), dimnames = dimnames(y.matrix))

##   yred     ygreen      yblue
## 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
```

The last are the estimated success probabilities $\hat{\pi} = \hat{\mu}$ because the sample size is one for each row.

Section 9 gives a formula for Fisher information for the saturated model.

```
fisher <- (Diagonal(x = mu) -
    mu %*% t(mu)) * modmat.strata %*% t(modmat.strata)
```

Then Fisher information for the canonical affine submodel is

```
modmat.foo <- modmat.covariates[ , ! is.na(beta.lcm), drop = FALSE]
fisher.lcm <- t(modmat.foo) %*% fisher %*% modmat.foo
fisher.lcm <- as.matrix(fisher.lcm)
```

And the standard errors are

```
fisher.lcm.inv <- solve(fisher.lcm)
se.lcm <- beta.lcm
se.lcm[! is.na(se.lcm)] <- sqrt(diag(fisher.lcm.inv))

coefs <- cbind(beta.lcm, gdor.lcm, se.lcm, beta.lcm / se.lcm, NA)
coefs[ , 5] <- 2 * pnorm(- abs(coefs[ , 4]))
colnames(coefs) <- c("Estimate", "GDOR", "Std. Error", "z value", "Pr(>|z|)")
printCoefmat(coefs)

##            Estimate      GDOR Std. Error z value
## yred             NA  1.933333         NA      NA
## x:yred           NA -0.096667         NA      NA
## ygreen    7.891179 -1.866667   6.038369  1.3068
## x:ygreen -0.121403  0.023333   0.091259 -1.3303
## yblue            NA -1.866667         NA      NA
## x:yblue          NA  0.023333         NA      NA
##          Pr(>|z|)
## yred           NA
## x:yred         NA
## ygreen     0.1913
## x:ygreen   0.1834
## yblue          NA
## x:yblue        NA
```

No real surprises. As always, the GDOR for the submodel canonical parameter is nearly impossible to interpret. We have to look at the corresponding η to understand (which we already did).

The P-values say the coefficients in the LCM are not statistically significantly different from zero, that is, the null hypothesis that blue and green do not depend on x and are 50-50 for all rows cannot be rejected with this small amount of data.

But these P-values do not say anything about the original model, that is, they do not say anything about red! If we wanted to test the null hypothesis that x does not effect any of the colors, then we have to fit that model.

```
gout.null <- glm.fit(modmat.strata, y, family = poisson())
coef(gout.null)

##      row1      row2      row3      row4      row5
## -1.098612 -1.098612 -1.098612 -1.098612 -1.098612
##      row6      row7      row8      row9
## -1.098612 -1.098612 -1.098612 -1.098612

gout.null$deviance

## [1] 19.77502
```

And, in order to have comparable log likelihoods, we have to refit the alternative hypothesis using the same R function and same data.

```
gout.alternative <- glm.fit(modmat, y, family = poisson())

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

gout.alternative$deviance

## [1] 4.955974
```

```
gout$deviance

## [1] 4.955974
```

(Actually, we see we didn't need this step.)

```
tstat <- gout.null$deviance - gout$deviance
pchisq(tstat, df = ncol(modmat) - ncol(modmat.strata),
    lower.tail = FALSE)

## [1] 0.02171179
```

So the effect of x is statistically significant. We cannot use the null model, and do need solutions at infinity.

F Big Data Example of Eck and Geyer

F.1 Data

```
rm(list = ls())
u <- "https://conservancy.umn.edu/bitstream/handle/11299/197369/bigcategorical.txt"
if (! file.exists("bigcategorical.txt"))
    download.file(u, "bigcategorical.txt")
foo <- read.table("bigcategorical.txt", header = TRUE)
```

F.2 Model Matrix and Response Vector

```
modmat <- sparse.model.matrix(y ~ (.)^4, data = foo)
dim(modmat)

## [1] 1024  781

y <- foo$y
```

F.3 Linear Programs

For once we follow Algorithm 1 exactly, including the loop. Except, we keep all of the optimal values found and all of the δ found, just to see what the algorithm does. In production code, we would not bother with that.

```
save.opt.val <- NULL
save.delta <- NULL
save.proc.time <- proc.time()

tangent.direction <- as.numeric(y == 0)
i.double.star <- rep(FALSE, length(y))
i.triple.star <- y == 0

objgrd <- rbind(tangent.direction) %*% modmat
objgrd <- as(objgrd, "numeric")

lp <- initProbGLPK()
addRowsGLPK(lp, nrow(modmat))

## [1] 1

addColsGLPK(lp, ncol(modmat))

## [1] 1

setSimplexParmGLPK(MSG_LEV, GLP_MSG_OFF) # STFU
# row bounds
lower.bounds <- pmin(- tangent.direction, 0)
upper.bounds <- pmax(- tangent.direction, 0)
idx <- which(lower.bounds == upper.bounds)
setRowsBndsGLPK(lp, idx, lower.bounds[idx],
    upper.bounds[idx], rep(GLP_FX, length(idx)))
idx <- which(lower.bounds != upper.bounds)
setRowsBndsGLPK(lp, idx, lower.bounds[idx],
    upper.bounds[idx], rep(GLP_DB, length(idx)))
# col bounds: free
idx <- 1:ncol(modmat)
setColsBndsGLPK(lp, idx, rep(-Inf, length(idx)),
    rep(Inf, length(idx)), rep(GLP_FR, length(idx)))
# objective function
```

```r
setObjDirGLPK(lp, GLP_MIN)
idx <- which(objgrd != 0)
setObjCoefsGLPK(lp, idx, objgrd[idx])
# constraint matrix (modmat)
foompter <- mat2triplet(modmat)
loadMatrixGLPK(lp, nnzero(modmat), foompter$i, foompter$j, foompter$x)

repeat {
    solveSimplexGLPK(lp)
    stopifnot(getPrimStatGLPK(lp) == GLP_FEAS)
    opt.val <- getObjValGLPK(lp)
    save.opt.val <- c(save.opt.val, opt.val)
    if (opt.val >= -0.001) break
    delta <- getColsPrimGLPK(lp)
    names(delta) <- colnames(modmat)
    save.delta <- rbind(save.delta, delta)
    eta <- getRowsPrimGLPK(lp)
    i.double.star <- i.double.star | abs(eta) > 0.001
    i.triple.star <- i.triple.star & (! i.double.star)
    if (! any(i.triple.star)) break
    objgrd <- rbind(i.triple.star * tangent.direction) %*% modmat
    objgrd <- as(objgrd, "numeric")
    idx <- 1:length(objgrd)
    setObjCoefsGLPK(lp, idx, objgrd)
    lower.bounds[(! i.triple.star) & (y == 0)] <- -Inf
    idx <- which(is.infinite(lower.bounds))
    setRowsBndsGLPK(lp, idx, lower.bounds[idx],
        upper.bounds[idx], rep(GLP_UP, length(idx)))
}

delProbGLPK(lp)

total.time <- proc.time() - save.proc.time
```

What happened?

```
save.opt.val

## [1] -68 -14   0
```

So we only had to do 3 linear programs, and it only took 0.175 of a second. This problem took several days to do according to the methods of Geyer (2009).

```
gamma <- colSums(save.delta)
eta <- modmat %*% gamma
eta <- as(eta, "numeric")

range(eta[abs(eta) <= 0.001])

## [1] -8.184212e-15  7.043408e-15

range(eta[abs(eta) >= 0.001])

## [1] -3 -1
```

We see that for this (big) problem the computer arithmetic is fairly accurate, much better than the results we got with R package `clpAPI` (now archived) But it is still good that we are using a sloppy tolerance to allow for any inaccuracy in computer arithmetic.

But, of course, slop is not rigor. So on to proofs.

F.4 Proof Level 1

```
is.zero <- abs(eta) <= 0.001
```

Thus the matrix Z in the linear equations (16) is

```
z <- modmat[is.zero, , drop = FALSE]
dim(z)

## [1] 942 781
```

We found doing this kind of proof before (Sections B.6 and E.7 above), that Z had redundant rows. So we use R function `redundant` in R package `rcdd` to eliminate them.

```
save.proc.time <- proc.time()
vrep <- makeV(lines = as.matrix(z))
rout <- redundant(vrep)
```

```
z <- z[rout$new.position > 0, , drop = FALSE]
dim(z)

## [1] 758 781

total.time <- proc.time() - save.proc.time
```

This the first code chunk in this document that takes so long, 2 minutes and 43.5 seconds, that we turn on caching for it.

```
save.proc.time <- proc.time()
z <- as.matrix(z)
z <- gmp::as.bigq(z)
a <- cbind2(diag(ncol(z)), t(z))
a <- rbind2(a, cbind2(z, 0 * diag(nrow(z))))
dim(a)

## [1] 1539 1539

b <- c(gamma, rep(0, nrow(z)))
b <- gmp::as.bigq(b)
length(b)

## [1] 1539

sout <- solve(a, b)
gamma <- sout[seq(along = gamma)]
modmat.gmp <- as(modmat, "matrix")
modmat.gmp <- gmp::as.bigq(modmat.gmp)
eta <- gmp::`%*%`(modmat.gmp, gamma)
eta <- as(eta, "numeric")
all(eta * tangent.direction <= 0)

## [1] TRUE

all(tangent.direction != 0 | eta == 0)

## [1] TRUE

identical(is.zero, eta == 0)

## [1] TRUE

total.time <- proc.time() - save.proc.time
```

Since this `eta` has the correct signs, we have proved the `gamma` it came from is a DOR. Hence the MLE for the OM does not exist.

And this code chunk took even more time, 18 minutes and 35.4 seconds, so we turn on caching for it too.

And the total time for this proof (both code chunks) was 21 minutes and 18.9 seconds, so we definitely want to make this optional. It's not something you want to do unless the problem is small (so it will be fast) or unless the problem is really important.

F.5 Proof Level 2

Now we do the linear program to prove that the DOR found in the preceding section is a GDOR. The objective function for this program is

```
objgrd <- colSums(modmat[y == 0 & is.zero, , drop = FALSE])
range(objgrd)

## [1]   0 287
```

We checked whether the objective function gradient was the zero vector in hopes of getting lucky, like in Section B.7 above, but no such luck here. So we proceed, like in Section E.8 above.

```
modmat.rcdd <- as(modmat, "matrix")
modmat.rcdd <- d2q(modmat.rcdd)
foompter <- modmat.rcdd[y == 0, , drop = FALSE]
hrep <- makeH(foompter, rep(0, nrow(foompter)))
foompter <- modmat.rcdd[y == 0 & eta == 0, , drop = FALSE]
hrep <- addHin(qneg(foompter), rep(1, nrow(foompter)), hrep)
foompter <- modmat.rcdd[y != 0, , drop = FALSE]
hrep <- addHeq(foompter, rep(0, nrow(foompter)), hrep)
```

```
save.proc.time <- proc.time()
lout <- lpcdd(hrep, d2q(objgrd))
lout$solution.type

## [1] "Optimal"

lout$optimal.value
```

```
## [1] "0"

total.time <- proc.time() - save.proc.time
```

That was fast, 3 minutes and 11.7 seconds, much faster than the level one proof (for this problem). But since there is no point to this proof, until the level one proof is done (this proof doesn't prove anything by itself, it only proves that the DOR found by the level one proof, if it did so, is a GDOR). So this method of proof has to be optional too. And even this much time is too long to be the default.

The total time for all proofs (both levels) was 24 minutes and 30.6 seconds. That's a lot of time to twiddle your thumbs, but Eck and Geyer (2021, Section 5.3) say doing this example using the methods of Geyer (2009) took a little over 3 days and 4 hours of computing time on a computer with clock speed of 3.4 GHz. So what counts as slow is relative.

For comparison, the computer on which this document was processed reports itself as `11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz`.

References

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

Agresti, A. (2022). R package `CatDataAnalysis`: Datasets for Categorical Data Analysis by Agresti, version 0.1-5. `https://CRAN.R-project.org/package=CatDataAnalysis`.

Axler, S. (2024). *Linear Algebra Done Right*, fourth edition. Springer, Cham.

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

Bates, D., and Maechler, M. (2024). R package `Matrix`: Sparse and Dense Matrix Classes and Methods, version 1.6-5. `https://CRAN.R-project.org/package=Matrix`.

Bloch, J. (2018). *Effective Java*, third edition. Addison-Wesley, Boston.

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

Burnham, K. P., and Anderson, D. R. (2002). *Model Selection and Multimodel Inference: A Practical Information-Theoretic Approach*, second edition. Springer, New York.

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. (2022). R package `glpkAPI`: R Interface to C API of GLPK, version 1.3.4. `https://CRAN.R-project.org/package=glpkAPI`.

Geyer, C. J. (1990). *Likelihood and Exponential Families*. PhD thesis, University of Washington. `https://hdl.handle.net/11299/56330`.

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. (2017). R package `aster2`: Aster Models, version 0.3. `http://cran.r-project.org/package=aster2`.

Geyer, C. J. (2023). R package `aster`: Aster Models, version 1.1-3. `http://cran.r-project.org/package=aster`.

Geyer, C. J. (2021b). Statistics 5102 (Geyer, Fall 2016) Examples: Coverage of Confidence Intervals. `https://www.stat.umn.edu/geyer/5102/examp/coverage.html`.

Geyer, C. J., and Johnson, L. T. (2023). R package `mcmc`: Markov Chain Monte Carlo, version 0.9-8. `https://CRAN.R-project.org/package=mcmc`.

Geyer, C. J., and Meeden, G. D. (2005). Fuzzy and randomized confidence intervals and P-values (with discussion). *Statistical Science*, **20**, 358–387. doi:10.1214/088342305000000340.

Geyer, C. J., Meeden, G. D., and Fukuda, K. (2023). R package `rcdd`: C Double Description for R, version 1.6. `http://CRAN.R-project.org/package=rcdd`

Geyer, C. J., Wagenius, S., and Shaw, R. G. (2007). Aster models for life history analysis. *Biometrika*, **94**, 415–426. doi:10.1093/biomet/asm030.

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

Hornik, K. (2020). R FAQ. https://CRAN.R-project.org/doc/FAQ/R-FAQ.html.

Konishi, S., and Kitagawa, G. (2008). *Information Criteria and Statistical Modeling*. Springer, New York.

Lang, S. (1993). *Real and Functional Analysis*, third edition. Springer-Verlag, New York.

Lucas, A., Scholz, I., Boehme, R., Jasson, S., and Maechler, M. (2024). R package `gmp`: Multiple Precision Arithmetic, version 0.7-4. https://CRAN.R-project.org/package=gmp.

Maechler, M. (2024). R package `sfsmisc`: Utilities from 'Seminar fuer Statistik' ETH Zurich, version 1.1-17. https://CRAN.R-project.org/package=sfsmisc.

R Core Team (2021). R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna. https://www.R-project.org/.

Rockafellar, R. T. (1970). *Convex Analysis*. Princeton University Press.

Rockafellar, R. T., and Wets, R. J.-B. (1998). *Variational Analysis*. Springer-Verlag, Berlin. (The corrected printings contain extensive changes. We used the third corrected printing, 2010.)

Varadhan, R. (2023). R package `alabama`: Constrained Nonlinear Optimization, version 2023.1.0. https://CRAN.R-project.org/package=alabama.

Wikipedia contributors (2021). Multiple comparisons problem. Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Multiple_comparisons_problem&oldid=1056327420. Accessed 26 November 2021 08:45 UTC.