Modeling Overdispersion in R

Andrew M. Raim*, Nagaraj K. Neerchal & Jorge G. Morel

Department of Mathematics and Statistics
University of Maryland, Baltimore County
Baltimore, MD 21250, U.S.A.

Contents

Document last modified: 2015-01-27 07:06
*Now with the U.S. Census Bureau
Email for correspondence: Andrew Raim (`araim1@umbc.edu`)

1 Introduction

The book *Overdispersion Models in SAS* by Morel and Neerchal (2012) discusses statistical analysis of categorical and count data which exhibit overdispersion, with a focus on computational procedures using SAS. This document retraces some of the ground covered in the book, which we abbreviate throughout as *OMSAS*, with the objective of carrying out similar analyses in R (R Core Team, 2014). Rather than attempting to cover every example in OMSAS, we will focus on two specific goals: analysis based on binomial/multinomial likelihoods which support extra variation, and model selection with the binomial goodness-of-fit (GOF) test. We will not cover examples based on count data, but extension to those should not be difficult. We will generally not spend much time discussing the data, on justification for the selected models, or on interpretation of the results. The reader should refer to OMSAS for more complete discussions of the examples and statistical models. In several places we will present additional material not found in OMSAS, such as the binomial finite mixture and the recently proposed Mixture Link binomial model.

We will assume that the reader has basic competency in R: that you can install and start R on your platform of choice, issue commands through the prompt, run script files, and write basic programs. Some discussion of R programming is given as we proceed in the document, but beginners may want to start with a more basic guide. For example, Rutgers University Libraries (`libguides.rutgers.edu/data_R`) offers a series of video tutorials which appears to be very instructive. We also recommend the RStudio program (`www.rstudio.com`) as a friendly user interface to R; it is free and available for all major platforms: Windows, Mac, and Linux.

Why consider R when there are many statistical computing packages available, including SAS? SAS and R are perhaps the most popular computing tools for statisticians and data analysts at the present time. Both have strengths and weaknesses, and neither is uniformly better for every application. SAS is commercial software which is developed and maintained by a corporation. Procedures generally work reliably and are well documented, but the software is costly. On the other hand, R is free and open source software, maintained by an active community of users. A wide variety of user-contributed packages are available, but their quality — documentation, reliability, and completeness — varies greatly as there may be little oversight. The programming models between SAS and R are also very different: R is built on a unified programming language, while SAS programs are composed of disjoint blocks for data manipulation (data steps), procedure calls, and numerical manipulation (e.g. `PROC IML`). Programming modes do not necessarily share a common syntax; they are often coordinated by macro programming, and results are passed between them via datasets.

Some of the analysis methods in this document are possible through base R, while some are possible through contributed packages.[1] We will largely avoid using contributed packages. Instead, we develop a numerical MLE framework that produces roughly the same results as those obtained in OMSAS using `PROC GLIMMIX` and `PROC NLMIXED`. The numerical MLE framework is quite flexible: the user provides a dataset and a likelihood, and the framework produces estimates, standard errors, p-values, confidence intervals, and more. To do this, we use basic statistical theory such as large sample Normality of the MLE and the delta method. We also make use of numerical optimization and numerical computation of a function's Jaco-

[1]Readers can check CRAN (`cran.r-project.org`), which is traditionally the major source of R packages, to get an idea of the large number of contributed packages that are available.

bian. Of course, by no means does this result in the best inference for every problem. It may give invalid results if used carelessly; for example, when assumptions about large sample size or independence are not met. Also, naively using numerical optimization on a complicated likelihood may not yield a good solution to the MLE problem. Still, the framework gives a very convenient way of carrying out an analysis whose result is often more or less equivalent to PROC GLIMMIX or PROC NLMIXED. Inference techniques that require derivation by hand for a given model cannot be programmed in such a general way. As an alternative to the general numerical optimization framework, we also briefly present a framework based on scoring, which includes Newton-Raphson, Fisher scoring, and several other algorithms as specific cases. Estimates, standard errors, p-values, etc can also be obtained from this scoring framework, although it is less convenient to the analyst as each model requires programming of quantities such as the score and information matrix. Bayesian statistical analysis is not discussed here or in OMSAS, but MCMC methods can also be programmed in a very general way. For example, the package LaplacesDemon (www.bayesian-inference.com/software) carries out MCMC sampling given data, a likelihood, a prior, and selection of an MCMC sampler.

Throughout this document, a box with a white background will represent an R prompt.

```
> cat("Hello world\n")
Hello world
```

The symbol ">" at the beginning of a line represents the prompt, and the text that follows is to be entered by the user. Lines that begin without ">" represent output. A box with a grey background represents a script or file.

```
cat("Hello world\n")
x <- y + z
```

The rest of the document proceeds as follows. Section 2 discusses distributions to be used in later examples and shows how to work with them in R. Section 3 presents the numerical MLE framework and shows its development piece-by-piece in R. Section 4 presents a scoring framework which may be used to compute MLEs without relying on a sophisticated optimization routine. In Section 5, a goodness-of-fit test for binomial data is presented along with an R implementation. Here, the parametric bootstrap is demonstrated to produce a specific p-value, as the basic test gives only an interval for the p-value. When we arrive at Section 6, we have the tools necessary to reproduce some of the data analyses from OMSAS. Readers may want to start here to see the finished product, then jump back to earlier sections for implementation details. Finally, Section 7 mentions some special considerations in R programming.

1.1 OverdispersionModelsInR Package

An R package called OverdispersionModelsInR is available for use with this document. It contains the full code for the distributions covered in this document, the numerical MLE framework, and the goodness-of-fit procedure. The reader can repeat the steps of the data analysis examples in Section 6 once the package is installed. The latest version of the package and installation instructions will be maintained at www.umbc.edu/~araim1/OverdispersionModelsInR. The reader may also wish to browse the source code in the package to see completed versions of the procedures discussed in this document.

1.2 Example Datasets

The datasets used in Section 6 have been adapted from ones which are freely available from the SAS website for OMSAS; see www.sas.com/store/prodBK_62693_en.html and follow the link "Example Code and Data". We have reorganized the data into a form which is more conveniently read into R. To err on the side of caution, we are not making these versions of the datasets publicly available without obtaining permission from the publisher of OMSAS. For now, this unfortunately means the reader must download the original examples from the SAS website, extract the data from the files, and put it into a more convenient form yourself. We suggest using comma-separated files as in the example below.

File: /home/araim/R/example.dat

```
Id,X,Y,Z
1,10,20,30
2,11,21,32
3,13,18,16
```

The data may then be read it into R as follows.

```
> dat <- read.table("/home/araim/R/example.dat", sep = ",", header = TRUE)
> dat
  Id  X  Y  Z
1  1 10 20 30
2  2 11 21 32
3  3 13 18 16
```

1.3 Tarball of Example Code

Some of the examples from this document have been made available in a tarball

OverdispersionModelsInR_examples.tar.gz

for the reader's convenience. This tarball should be available alongside the document. It includes many of the longer examples which may be tedious to retype.

2 Distributions

R features support for a number of commonly encountered probability distributions. A standard pattern is followed in the implementation of these distributions which consists of four functions. For example, the Normal distribution is implemented in the functions:

- dnorm computes the density,
- rnorm draws random numbers from the distribution,
- pnorm computes the cumulative distribution function,
- qnorm computes the quantiles.

The binomial distribution has similar functions, but with the norm suffix replaced by binom. To see a list of the built-in distributions, the command ?Distributions brings up the help page.

```
> ?Distributions

Distributions             package:stats              R Documentation

Distributions in the stats package

Description:

     Density, cumulative distribution function, quantile function and
     random variate generation for many standard probability
     distributions are available in the 'stats' package.

Details:

     The functions for the density/mass function, cumulative
     distribution function, quantile function and random variate
     generation are named in the form 'dxxxv, 'pxxx', 'qxxx' and 'rxxx'
     respectively.

     For the beta distribution see 'dbeta'.

     For the binomial (including Bernoulli) distribution see 'dbinom'.

     For the Cauchy distribution see 'dcauchy'.
...
```

To see the help page for any properly documented R function, enter ? followed by the function name.

```
> ?dnorm
Normal                  package:stats                  R Documentation

The Normal Distribution

Description:

     Density, distribution function, quantile function and random
     generation for the normal distribution with mean equal to 'mean'
     and standard deviation equal to 'sd'.

Usage:

     dnorm(x, mean = 0, sd = 1, log = FALSE)
     pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
...
```

Note that the functions `dnorm`, `rnorm`, `pnorm`, and `qnorm` share a common help page.

2.1 Normal

Here is a quick demonstration of the Normal distribution. First we will compute the density of N$(1, 5^2)$, the Normal distribution with mean 1 and variance 5^2.

```
> dnorm(1.3, mean = 1, sd = 5)
[1] 0.07964497
```

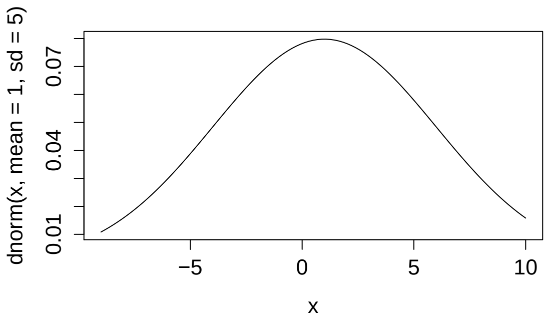
The density is a "vectorized" function, which means that it can be used to evaluate many points at once.

```
> y <- seq(-5, 5, 0.25)
> y
 [1] -5.00 -4.75 -4.50 -4.25 -4.00 -3.75 -3.50 -3.25 -3.00 -2.75 -2.50 -2.25 -2.00 -1.75
[15] -1.50 -1.25 -1.00 -0.75 -0.50 -0.25  0.00  0.25  0.50  0.75  1.00  1.25  1.50  1.75
[29]  2.00  2.25  2.50  2.75  3.00  3.25  3.50  3.75  4.00  4.25  4.50  4.75  5.00
```

```
> dnorm(y, mean = 1, sd = 5)
 [1] 0.03883721 0.04118725 0.04357044 0.04597643 0.04839414 0.05081181 0.05321705 0.05559698
 [9] 0.05793831 0.06022749 0.06245079 0.06459447 0.06664492 0.06858877 0.07041307 0.07210539
[17] 0.07365403 0.07504807 0.07627756 0.07733362 0.07820854 0.07889587 0.07939051 0.07968878
[25] 0.07978846 0.07968878 0.07939051 0.07889587 0.07820854 0.07733362 0.07627756 0.07504807
[33] 0.07365403 0.07210539 0.07041307 0.06858877 0.06664492 0.06459447 0.06245079 0.06022749
[41] 0.05793831
```

We can plot the density of N$(1, 5^2)$ using the `curve` function. In the following command, `x` is a dummy variable used by the `curve` command, and not a variable that has previously been given a value. It produces the following plot.

```
> curve(dnorm(x, mean = 1, sd = 5), xlim = c(-9, 10))
```



We can also compute the density ourselves using basic arithmetic.

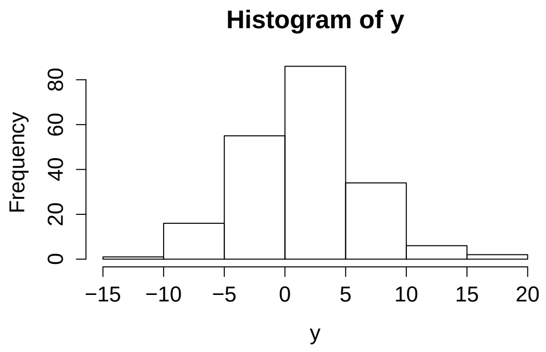
```
> mu <- 1
> sigma <- 5
> y <- seq(-5, 5, 0.25)
> (2 * pi * sigma^2)^(-1/2) * exp( -1/2 * (y - mu)^2 / sigma^2 )
 [1] 0.03883721 0.04118725 0.04357044 0.04597643 0.04839414 0.05081181 0.05321705 0.05559698
 [9] 0.05793831 0.06022749 0.06245079 0.06459447 0.06664492 0.06858877 0.07041307 0.07210539
[17] 0.07365403 0.07504807 0.07627756 0.07733362 0.07820854 0.07889587 0.07939051 0.07968878
[25] 0.07978846 0.07968878 0.07939051 0.07889587 0.07820854 0.07733362 0.07627756 0.07504807
[33] 0.07365403 0.07210539 0.07041307 0.06858877 0.06664492 0.06459447 0.06245079 0.06022749
[41] 0.05793831
```

Next we will draw some random numbers from N$(1, 5^2)$.

```
> y <- rnorm(20, mean = 1, sd = 5)
> y
 [1] -2.7226253  3.2208362 10.1373916  2.9912574  2.8425974 -0.9036215 10.0942682 -0.8051708
 [9] -1.5532178 -3.8122432  5.5623794  9.6967335 -4.0715477  5.3016421  5.3031682 -1.7415364
[17]  2.6894637 -0.4759012 11.4477871  7.0996068
```

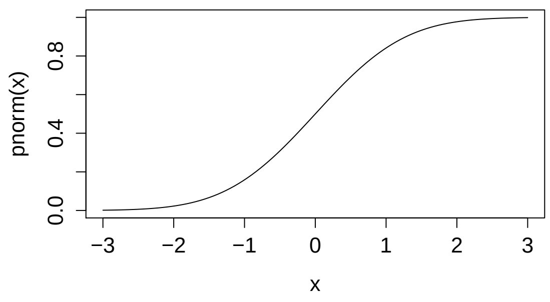
This represents an iid (independent and identically distributed) sample, but it is also possible to draw an independent-but-not-identically-distributed sample by specifying a vector for `mean` and/or `sd`. Let us take a larger iid sample and plot the histogram.

```
> y <- rnorm(200, mean = 1, sd = 5)
> hist(y)
```



For completeness, examples of the quantile and cumulative distribution functions are given below. Notice that if we leave out the `mean` and `sd` arguments, the functions default to mean 0 and sd 1, which represents the standard Normal distribution.

```
> qnorm(1 - 0.025)
[1] 1.959964
> pnorm(0)
[1] 0.5
> curve(pnorm(x), xlim = c(-3, 3))
```



2.2 Binomial

Next we will consider the binomial distribution. We will denote Bin(m, p) as the binomial distribution with m trials with probability of success p. Recall that if X follows this distribution,

$$\mathrm{E}(X) = mp, \quad \mathsf{Var}(X) = mp(1 - p).$$

The `dbinom` function is available for computing the density. Notice that a warning is produced if we attempt to evaluate the density on a number outside of the sample space.

```
> dbinom(10, size = 20, prob = 1/4)
[1] 0.009922275
> dbinom(10.5, size = 20, prob = 1/4)
[1] 0
Warning message:
In dbinom(10.5, size = 20, prob = 1/4) : non-integer x = 10.500000
> y <- 0:20
> dbinom(y, size = 20, prob = 1/4)
 [1] 3.171212e-03 2.114141e-02 6.694781e-02 1.338956e-01 1.896855e-01 2.023312e-01 1.686093e-01
 [8] 1.124062e-01 6.088669e-02 2.706075e-02 9.922275e-03 3.006750e-03 7.516875e-04 1.541923e-04
[15] 2.569872e-05 3.426496e-06 3.569266e-07 2.799425e-08 1.555236e-09 5.456968e-11 9.094947e-13
```

We can also compute the density ourselves. Recall that density is in the form $\binom{m}{x}p^x(1-p)^{m-x}$. The $\binom{m}{x}$ portion can produce very large numbers while the $p^x(1-p)^{m-x}$ can produce very small numbers. Computation of the density is more stable if done at the logarithmic scale and then translated back to the probability scale. The function $\log(x!)$ can be computed as `lgamma(x + 1)`, where `lgamma` calculates the logarithm of the Γ function.

```
> m <- 20
> p <- 1/4
> log.ff <- lgamma(m+1) - lgamma(y+1) - lgamma(m-y+1) + y*log(p) + (m-y)*log(1-p)
> exp(log.ff)
 [1] 3.171212e-03 2.114141e-02 6.694781e-02 1.338956e-01 1.896855e-01 2.023312e-01 1.686093e-01
 [8] 1.124062e-01 6.088669e-02 2.706075e-02 9.922275e-03 3.006750e-03 7.516875e-04 1.541923e-04
[15] 2.569872e-05 3.426496e-06 3.569266e-07 2.799425e-08 1.555236e-09 5.456968e-11 9.094947e-13
```

We can draw from the binomial distribution using the built-in `rbinom` function, or code the procedure ourselves, noting that binomial is a sum of independent Bernoulli trials. To write the code ourselves, we can use the `sample` function to draw 0 or 1 with replacement, then take the sum. We can do this in a loop to obtain the desired number of binomial draws. Note that loops in R are generally not an efficient way to program, see Section 7.

```
y <- integer(10)
for (i in 1:10)
{
    z <- sample(x = c(1,0), size = 20, replace = TRUE, prob = c(1/4, 3/4))
    y[i] <- sum(z)
}
```

```
> rbinom(10, size = 20, prob = 1/4)
 [1] 3 2 8 3 6 6 3 1 3 9
> y
 [1] 3 2 4 3 5 7 2 7 5 5
```

2.3 Beta-Binomial

While not available as one of the standard distributions in R, beta-binomial (BB, see OMSAS section 4.2) has been implemented in several packages. However, we will write the code ourselves for the sake of demonstration. First, let us recall that the distribution is obtained by assuming that the binomial probability of success is randomly drawn from the beta distribution,

$$\begin{aligned} Y \mid \mu &\sim \text{Bin}(m, \mu), \\ \mu &\sim \text{Beta}(\alpha, \beta). \end{aligned}$$

Here, Beta(α, β) denotes the beta distribution with density $f(x) = [B(\alpha,\beta)]^{-1}x^{\alpha-1}(1-x)^{\beta-1}$ on the interval $(0, 1)$, where $B(\alpha,\beta) = \Gamma(\alpha)\Gamma(\beta)/\Gamma(\alpha+\beta)$. The marginal density of the observed data are then given by

$$f(x \mid \alpha, \beta) = \binom{m}{x}\frac{B(\alpha + x, \beta + m - x)}{B(\alpha, \beta)}, \quad x \in \{0, \dots, m\}.$$

BB may be reparameterized, as noted in OMSAS, using

$$\begin{aligned} &\alpha = \pi\rho^{-1}(1-\rho) \quad \text{and} \quad \beta = (1-\pi)\rho^{-1}(1-\rho) \\ \iff &\pi = \frac{\alpha}{\alpha+\beta} \quad \text{and} \quad \rho = \frac{1}{\alpha+\beta+1}, \end{aligned}$$

so that $\pi = \mathrm{E}(Y/m)$ can be interpreted as the probability of success. The parameter ρ captures the amount of overdispersion, in the sense that

$$\mathsf{Var}(Y) = m\pi(1-\pi)\{1+\rho(m-1)\}.$$

As $\rho \downarrow 0$, BB approaches the standard binomial distribution. Note that both π and ρ must be in the unit interval $(0, 1)$.

Let us create a file called `betabin.R` which will compute the density and produce draws from the beta-binomial distribution.

File: /home/araim/R/betabin.R

```
d.beta.binom <- function(x, Pi, rho, m, log = FALSE)
{
    a <- Pi * rho^(-2) * (1 - rho^2)
    b <- (1-Pi) * rho^(-2) * (1 - rho^2)
    log.ff <- lgamma(m+1) - lgamma(x+1) - lgamma(m-x+1) + lgamma(a+x) +
        lgamma(b+m-x) - lgamma(a+b+m) + lgamma(a+b) - lgamma(a) - lgamma(b)

    if (log) return(log.ff)
    else return(exp(log.ff))
}

r.beta.binom <- function(n, Pi, rho, m)
{
    a <- Pi * rho^(-2) * (1 - rho^2)
    b <- (1-Pi) * rho^(-2) * (1 - rho^2)

    z <- rbeta(n, a, b)
    rbinom(n, size = m, prob = z)
}
```

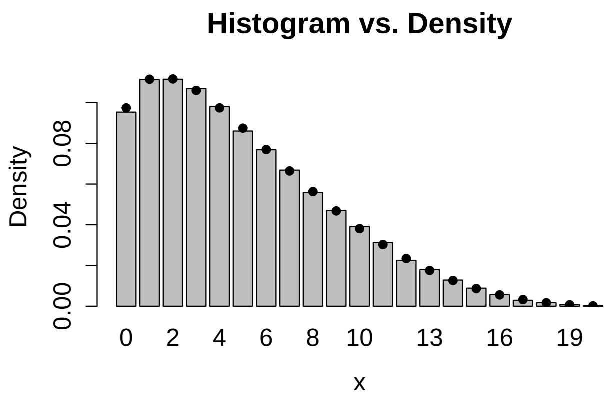
Notice that the statement `log = FALSE` is given in the argument list so that `log` will be `FALSE` by default, and the returned result will be given on the probability scale. In this example, the code file has been placed at the location `/home/araim/R` on a Linux computer. Mac paths will be similar. Windows users can use paths like `C:\\Users\\araim\\R` (where each backslash character is escaped by another backslash), or `C:/Users/araim/R` (with a forward slash). We can now read the `R` script `/home/araim/R/betabin.R` into our session to define our functions so that we can use them.

```
> source("/home/araim/R/betabin.R")
> d.beta.binom(x = 10, Pi = 1/4, rho = 0.2, m = 20)
[1] 0.02637418
> y <- 0:20
> d.beta.binom(y, Pi = 1/4, rho = 0.2, m = 20)
 [1] 1.655831e-02 5.370264e-02 9.920071e-02 1.360467e-01 1.530525e-01 1.484146e-01 1.275438e-01
 [8] 9.874356e-02 6.953193e-02 4.475618e-02 2.637418e-02 1.420831e-02 6.967538e-03 3.087155e-03
[15] 1.221999e-03 4.250431e-04 1.267884e-04 3.125317e-05 5.990191e-06 7.964796e-07 5.531109e-08
> d.beta.binom(y, Pi = 1/4, rho = 0.2, m = 20, log = TRUE)
 [1]  -4.100867  -2.924293  -2.310610  -1.994757  -1.876974  -1.907746  -2.059296  -2.315229
 [9]  -2.665969  -3.106526  -3.635370  -4.253928  -4.966493  -5.780505  -6.707267  -7.763320
[17]  -8.972991 -10.373390 -12.025387 -14.043064 -16.710292
> r.beta.binom(n = 10, Pi = 1/4, rho = 0.2, m = 20)
 [1] 4 4 2 3 3 5 6 6 3 4
```

As a quick test to ensure that the two functions are correct (at least that they implement the same distribution), consider comparing the histogram of an iid sample and the density itself. As the sample size becomes large, the histogram should become close to the density.

```r
n <- 100000; m <- 20
Pi <- 1/4; rho <- 0.4
x <- 0:m
y <- r.beta.binom(n, Pi, rho, m)
f <- factor(y, levels=0:m)
p <- table(f)/n
ylim <- c(0, max(p) * 1.05)
coords <- barplot(p, ylim = ylim, xlab = "x", ylab = "Density")
points(coords, d.beta.binom(x, Pi, rho, m), pch = 19)
title("Histogram vs. Density")
```

Note that because the data are discrete, the function `barplot` is used here to display the counts at each point in the sample space. The `hist` function could be used instead to bin the data into intervals. `barplot` returns `coords`, which help us to plot the density at each bar in a subsequent call. The `table` function is used to compute the frequency at each point in the sample space; we divide the frequencies by n to obtain propotions. There is a chance that some of the values in the sample space will not be present in our sample; to ensure that these zero counts are present in our table, we explicitly convert `y` into a factor with values 0 to 20. Some adjustments are made to the plot to adjust the y-axis, add a title, and add axis labels.



2.4 Random-Clumped Binomial

The random-clumped binomial (RCB) distribution (see OMSAS section 4.3) provides an alternative to BB as a binomial model with extra variation. Denote Ber(p) as the Bernoulli distribution with probability of success p. RCB arises by considering variables

$$\begin{aligned} Y &\sim \text{Ber}(\pi), \\ X &\sim \text{Bin}(m - N, \pi), \\ N &\sim \text{Bin}(m, \rho). \end{aligned}$$

The observation $T = NY + X$ follows the RCB distribution with parameters π and ρ, notated as $T \sim \text{RCB}(m, \pi, \rho)$. Here, Y represents success/failure of a leader, N is the number of trials that follows the leader, and X represents the remaining trials that are selected independently. Therefore, although T is a sum of Bernoulli trials, the trials are not independent; the degree of depedence is increased as $\rho \uparrow 1$ and decreased as $\rho \downarrow 0$. We may interpret $\pi \in (0, 1)$ as an overall success probability for the RCB trials, and $\rho \in (0, 1)$ as the probability of following the leader. The density of RCB can be expressed as a finite mixture of two binomial densities (see Section 2.5),

$$f(t \mid m, \pi, \rho) = \pi \text{Bin}\Big(m, (1-\rho)\pi + \rho\Big) + (1-\pi)\text{Bin}\Big(m, (1-\rho)\pi\Big).$$

Notice that probability of success for each density in the mixture is a function of π and ρ. Therefore, we can think of RCB as a binomial finite mixture with some constraints on the parameters. The expectation and variance of T are the same as in the formulation of beta-binomial given in Section 2.3; namely,

$$\text{E}(T) = m\pi, \quad \text{Var}(T) = m\pi(1-\pi)\{1 + \rho(m-1)\}.$$

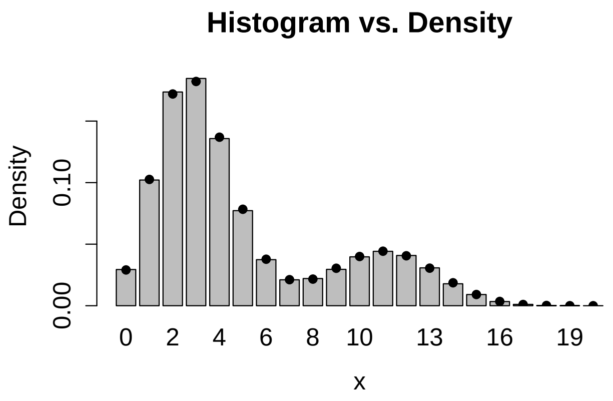
Here are functions to compute and draw from the RCB density.

File: /home/araim/R/rcb.R

```r
r.rcb <- function(n, Pi, rho, m)
{
    stopifnot( 0 < rho && rho < 1 )
    Y <- rbinom(n, prob = Pi, size = 1)
    N <- rbinom(n, prob = rho, size = m)
    X <- rbinom(n, prob = Pi, size = m-N)
    N*Y + X
}

d.rcb <- function(x, Pi, rho, m, log = FALSE)
{
    stopifnot( 0 < rho && rho < 1 )
    fc1 <- dbinom(x, prob = (1 - rho) * Pi + rho, size = m)
    fc2 <- dbinom(x, prob = (1 - rho) * Pi, size = m)
    ff <- Pi * fc1 + (1-Pi) * fc2

    if (log) return(log(ff))
    else return(ff)
}
```

Again, as a way to quickly check our code, we will compare the histogram of a very large iid sample to the density.

```r
source("/home/araim/R/rcb.R")
n <- 100000; m <- 20
Pi <- 1/4; rho = 0.4
x <- 0:m
y <- r.rcb(n, Pi, rho, m)
f <- factor(y, levels=0:m)
p <- table(f)/n
ylim <- c(0, max(p) * 1.05)
coords <- barplot(p, ylim = ylim, xlab = "x", ylab = "Density")
points(coords, d.rcb(x, Pi, rho, m), pch = 19)
title("Histogram vs. Density")
```



2.5 Binomial Finite Mixture

A finite mixture is formulated by taking a linear combination of J densities with corresponding weights $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_J)$. Many options are possible; for example, a zero-inflated binomial distribution can be obtained by mixing a binomial density with a point mass at zero. Here we consider the binomial finite mixture, where J binomial densities are mixed together. This is useful to represent a heterogeneous population with differing probabilities of success, or more generally just to obtain a model which can account for more variability than a single binomial density. A common issue is finding an appropriate choice of J. The binomial finite mixture has density

$$f(y \mid m, \boldsymbol{p}, \boldsymbol{\pi}) = \sum_{j=1}^{J} \pi_j \binom{m}{y} p_j^y (1 - p_j)^{m-y},$$

where $\boldsymbol{p} = (p_1, \ldots, p_J)$ and $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_J)$. We will use the notation $Y \sim \text{BinMix}_J(m, \boldsymbol{p}, \boldsymbol{\pi})$ to describe a random variable with this distribution. Note that

$$\mathrm{E}(Y) = m\boldsymbol{\pi}^T\boldsymbol{p}, \quad \text{Var}(Y) = m(m-1)\sum_{j=1}^{J} \pi_j p_j^2 + m\boldsymbol{\pi}^T\boldsymbol{p}\Big(1 - m\boldsymbol{\pi}^T\boldsymbol{p}\Big)$$

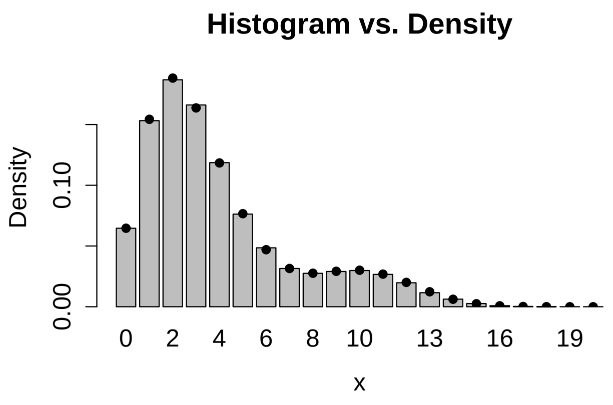
Let us create a file `/home/araim/R/binmix.R` with functions to compute and draw from this density.

File: /home/araim/R/binmix.R

```
d.binom.mix <- function(x, p, Pi, m, log = FALSE)
{
    J <- length(Pi)
    fc <- matrix(NA, length(x), J)
    for (j in 1:J) {
        fc[,j] <- dbinom(x, prob = p[j], size = m)
    }

    if (log) log(fc %*% Pi)
    else fc %*% Pi
}

r.binom.mix <- function(n, p, Pi, m)
{
    J <- length(Pi)
    z <- sample(1:J, prob = Pi, replace = TRUE, size = n)

    if (class(p) == "matrix") {
        idx <- cbind(1:n, z)
        prob <- p[idx]
    } else {
        prob <- p[z]
    }

    rbinom(n, prob = prob, size = m)
}
```

In the function `r.binom.mix`, we allow for the possibility that `p` is an `n x J` matrix, with the `i`th row corresponding to the `i`th observation, and the `J` columns in that row corresponding to its mixing probabilities. Here again is our test comparing the histogram of a large sample with the density.

```
source("/home/araim/R/binmix.R")
n <- 100000; m <- 20
x <- 0:m
p <- c(1/2, 1/5, 1/10)
Pi <- c(1/6, 2/6, 3/6)
y <- r.binom.mix(n, p, Pi, m)
f <- factor(y, levels=0:m)
prop <- table(f)/n
ylim <- c(0, max(prop) * 1.05)
coords <- barplot(prop, ylim = ylim, xlab = "x", ylab = "Density")
points(coords, d.binom.mix(x, p, Pi, m), pch = 19)
title("Histogram vs. Density")
```



2.6 Mixture Link Binomial

The Mixture Link model was introduced in (Raim, 2014), as a method to link the mean of a binomial finite mixture to a regression. This might be desirable when the probability of success for the population is the quantity of interest, but the population is heterogeneous or the data are otherwise fitted well by a finite mixture.

To formulate the Mixture Link binomial model, we first assume a binomial finite mixture

$$f(y \mid m, \boldsymbol{\theta}) = \sum_{j=1}^{J} \pi_j \binom{m}{y} \mu_j^t (1-\mu_j)^{m-t}$$

whose mean is $\mathrm{E}(Y) = m \sum_{j=1}^{J} \pi_j \mu_j$. The objective is to link a regression $\boldsymbol{x}^T\boldsymbol{\beta}$ to the mixed probability of success $\sum_{j=1}^{J} \pi_j \mu_j$ through an inverse link function G (say, the logistic CDF which is used for logistic regression). For a fixed $\boldsymbol{\beta}$ and $\boldsymbol{\pi}$, and a covariate $\boldsymbol{x}$, consider the set

$$A(\boldsymbol{x}, \boldsymbol{\beta}, \boldsymbol{\pi}) = \{\boldsymbol{\mu} \in [0,1]^J : \boldsymbol{\mu}^T \boldsymbol{\pi} = G(\boldsymbol{x}^T\boldsymbol{\beta})\},$$

which explicitly represents all the vectors of binomial probabilities $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_J)$ where the link is enforced. The Mixture Link model assumes $\boldsymbol{\mu}_i$ are subject-specific random effects drawn from the set $A(\boldsymbol{x}_i, \boldsymbol{\beta}, \boldsymbol{\pi})$ for $i = 1, \ldots, n$. The random effects are assumed to follow a transformation of the Dirichlet distribution, using the vertices of the polytope $A(\boldsymbol{x}_i, \boldsymbol{\beta}, \boldsymbol{\pi})$. The Mixture Link density is obtained by integrating out the random effects. The model can be written hierarchically as

$$\begin{aligned}
&Y_i \mid \boldsymbol{\mu}_i, \boldsymbol{\pi} \overset{\text{ind}}{\sim} \text{BinMix}(m_i, \boldsymbol{\mu}_i, \boldsymbol{\pi}) \\
&\boldsymbol{\mu}_i = \boldsymbol{V}^{(i)} \boldsymbol{\lambda}^{(i)}, \quad \text{where } \boldsymbol{V}^{(i)} = (\boldsymbol{v}_1^{(i)} \cdots \boldsymbol{v}_{k_i}^{(i)}) \text{ are vertices of } A(\boldsymbol{x}_i, \boldsymbol{\beta}, \boldsymbol{\pi}) \\
&\boldsymbol{\lambda}^{(i)} \overset{\text{ind}}{\sim} \text{Dirichlet}_{k_i}(\kappa, \ldots, \kappa),
\end{aligned}$$

and is parameterized by $\boldsymbol{\theta} = (\boldsymbol{\beta}, \boldsymbol{\pi}, \kappa)$. It may also be formulated when there is no regression, but the parameter of interest is the population probability of success $p = \sum_{j=1}^{J} \pi_j \mu_j$. In this case, the set representing enforcement the link is

$$A(p, \boldsymbol{\pi}) = \{\boldsymbol{\mu} \in [0,1]^J : \boldsymbol{\mu}^T \boldsymbol{\pi} = p\},$$

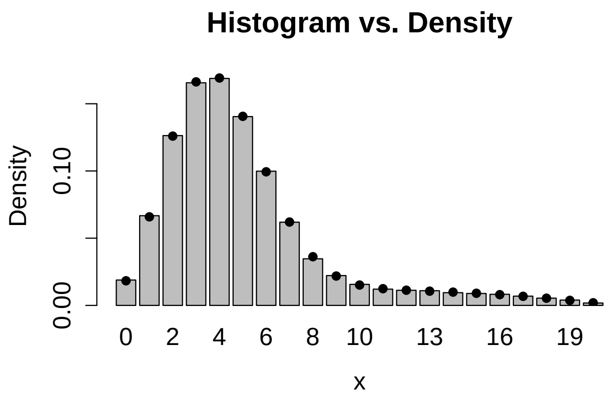
and the model is parameterized by $\boldsymbol{\theta} = (p, \boldsymbol{\pi}, \kappa)$. Denote $\boldsymbol{V} = (\boldsymbol{v}_1 \cdots \boldsymbol{v}_k)$ as the matrix whose columns are the vertices of $A(p, \boldsymbol{\pi})$. In the case of no regression, the expectation and variance can be written as

$$\mathrm{E}(Y) = mp, \quad \text{Var}(Y) = mp\,(1 - mp) + m(m-1) \sum_{j=1}^{J} \pi_j \frac{\boldsymbol{v}_{j\cdot}^T \boldsymbol{v}_{j\cdot} + \kappa (k \bar{v}_{j\cdot})^2}{k(1+\kappa k)},$$

where $\boldsymbol{v}_{j\cdot}$ denotes the jth row of the matrix $\boldsymbol{V}$, and $\bar{v}_{j\cdot}$ denotes the mean of that row.

Functions to evaluate and draw from the Mixture Link density are not included in this document because of their length and relative complexity compared to the previous distributions, but the reader may refer to the source code we have provided. Let us carry out the test comparing the histogram to the density, at least to demonstrate the functions.

```
n <- 100000; m <- 20
x <- 0:m
p <- 1/4; Pi <- c(1/6, 5/6); kappa <- 2
y <- r.mixture.link(n, p, Pi, kappa, m)
f <- factor(y, levels=0:m)
prop <- table(f)/n
ylim <- c(0, max(prop) * 1.05)
coords <- barplot(prop, ylim = ylim, xlab = "x", ylab = "Density")
points(coords, d.mixture.link(x, m, p, Pi, kappa), pch = 19)
title("Histogram vs. Density")
```



2.7 Multinomial Finite Mixture

Multinomial distributions are appropriate for trials with more than two possible outcomes. First consider the simple multinomial distribution involving m independent trials with k possible outcomes. Suppose the probabilities of the outcomes are given by $p_1, \ldots, p_k$. The density is given by

$$f(\boldsymbol{y} \mid m, \boldsymbol{p}) = \frac{m!}{y_1! \cdots y_k!} p_j^{y_1} \cdots p_k^{y_k}, \quad \boldsymbol{y} \in \Big\{(y_1, \ldots, y_k) : y_j \in \{0, \ldots, m\}, \sum_{j=1}^k y_j = m\Big\}.$$

A multinomial finite mixture is then contructed by mixing J multinomial densities,

$$f(\boldsymbol{y} \mid m, \boldsymbol{P}, \boldsymbol{\pi}) = \sum_{\ell=1}^J \pi_j \frac{m!}{y_1! \cdots y_k!} p_{\ell 1}^{y_1} \cdots p_{\ell k}^{y_k},$$

where $\boldsymbol{P} = (\boldsymbol{p}_1 \cdots \boldsymbol{p}_J)$ is a $k \times J$ matrix whose columns are probability vectors. We will use the notation $\boldsymbol{Y} \sim \text{MultMix}_J(m, \boldsymbol{P}, \boldsymbol{\pi})$ to describe a random variable with this distribution. We can write the expectation and variance of $\boldsymbol{Y}$ compactly as

$$\mathrm{E}(\boldsymbol{Y}) = \boldsymbol{P}\boldsymbol{\pi}, \quad \text{Var}(\boldsymbol{Y}) = m(m-1)\boldsymbol{P}\,\text{Diag}(\boldsymbol{\pi})\boldsymbol{P}^T + m\,\text{Diag}(\boldsymbol{P}\boldsymbol{\pi}) - m^2\boldsymbol{P}\boldsymbol{\pi}\boldsymbol{\pi}^T\boldsymbol{P}^T.$$

The notation Diag($\boldsymbol{x}$) represents a diagonal matrix constructed from $\boldsymbol{x} = (x_1, \ldots, x_n)$, with x_1 in the 1st row/1st column, x_2 in the 2nd row/2nd column, and so on.

The multinomial density function `dmultinom` in base `R` is not vectorized, so it is instructive (and useful) to write a version ourselves. The following version will be efficient if evaluated over many observations at once, but not when there are few observations with many categories. In the latter case, performance will be similar to calling the `dmultinom` in base `R` using a loop.

File: /home/araim/R/multinom.R

```r
dmultinom <- function(x, size, prob, log = FALSE)
{
    k <- nrow(x)
    n <- ncol(x)

    fj <- matrix(NA, n, k)
    for (j in 1:k) {
        fj[,j] <- -lgamma(x[j,]+1) + x[j,] * log(prob[j])
    }

    log.ff <- lgamma(m+1) + rowSums(fj)
    if (log) return(log.ff)
    else return(exp(log.ff))
}
```

Next we will implement the multinomial mixture density.

File: /home/araim/R/multmix.R

```r
r.multmix <- function(n, P, Pi, m)
{
    J <- length(Pi)
    k <- nrow(P)
    z <- sample(1:J, prob = Pi, replace = TRUE, size = n)

    if (length(m) == 1) m <- rep(m,n)
    y <- matrix(NA, nrow = k, ncol = n)

    for (i in 1:n) {
        y[,i] <- rmultinom(1, prob = P[,z[i]], size = m[i])
    }

    return(y)
}

d.multmix <- function(x, P, Pi, m, log = FALSE)
{
    Pi <- normalize(Pi)
    J <- length(Pi)
    n <- ncol(x)

    ff <- matrix(NA, n, J)
    for(j in 1:J) {
        ff[,j] <- dmultinom(x, m, P[,j])
    }

    res <- as.numeric(ff %*% Pi)
    if (log) return(log(res))
    else return(res)
}
```

Here, `P` is a `k x J` matrix. Again, we check the code by comparing the histogram of a large iid sample with the density. We elect to view the density and sample proportion as a table of discrete values rather than a simple plot. (It is also possible to plot each quantity on a two-dimensional grid if desired).

```
n <- 1000000
m <- 20

P <- matrix(NA, nrow = 3, ncol = 3)
P[,1] <- c(1/2, 1/4, 1/4)
P[,2] <- c(1/5, 1/10, 7/10)
P[,3] <- c(1/10, 5/10, 4/10)

Pi <- c(1/6, 2/6, 3/6)
y <- r.multmix(n, P, Pi, m)

for (i in seq(1,m+1)) {
    for (j in seq(1,m+1)) {
        x <- matrix( c(i-1, j-1, m - (i-1) - (j-1)), 3, 1 )
        if (x[3,] > m || x[3,] < 0) next
        cat("x =", x)
        printf("\tdensity = %g, ", d.multmix(x, P, Pi, m))
        printf("prop = %g\n", sum(y[1,] == x[1,] & y[2,] == x[2,]) / n)
    }
}
```

```
x = 0 0 20        density = 0.00026598, prop = 0.000277
x = 0 1 19        density = 0.000760064, prop = 0.000721
x = 0 2 18        density = 0.00103296, prop = 0.001054
x = 0 3 17        density = 0.000896237, prop = 0.000898
x = 0 4 16        density = 0.000601741, prop = 0.000604
...
```

The `printf` function is not built into R; we will define it later in Section 3.

2.8 Dirichlet-Multinomial

Dirichlet-multinomial (DM) is the multinomial analogue of the beta-binomial distribution. See Section 7.7 of OMSAS. In the DM distribution, the vector of category probabilities is assumed to be drawn from the Dirichlet distribution

$$\begin{aligned} \boldsymbol{Y} \mid \boldsymbol{\mu} &\sim \text{Mult}_J(m, \boldsymbol{\mu}), \\ \boldsymbol{\mu} &\sim \text{Dirichlet}_J(\boldsymbol{\alpha}). \end{aligned}$$

This makes DM one type of multinomial distribution with extra variation. We consider the parameterization which takes $\boldsymbol{\alpha} = c\boldsymbol{\pi}$, where $\boldsymbol{\pi}$ is a probability vector and $c = \rho^{-2}(1-\rho^2)$. The density is given by

$$f(\boldsymbol{y} \mid m, \boldsymbol{\pi}, \rho) = \frac{m!}{y_1! \cdots y_k!} \frac{\Gamma(c)}{\Gamma(c+m)} \prod_{j=1}^{k} \frac{\Gamma(y_j + c\pi_j)}{\Gamma(c\pi_j)},$$

and the notation $\boldsymbol{Y} \sim \text{DM}_k(m, \boldsymbol{\pi}, \rho)$ is used to describe a random variable with this density. The expectation and variance of $\boldsymbol{Y}$ are given by

$$\text{E}(\boldsymbol{Y}) = m\boldsymbol{\pi}, \quad \text{Var}(\boldsymbol{Y}) = m[1 + \rho^2(m-1)] \left[\text{Diag}(\boldsymbol{\pi}) - \boldsymbol{\pi}\boldsymbol{\pi}^T\right].$$

The following code allows us to evaluate and draw from the density in R.

File: /home/araim/R/dm.R

```r
r.dm <- function(n, Pi, rho, m)
{
    k <- length(Pi)
    if (length(m) == 1) m <- rep(m, n)
    C <- rho^(-2) * (1 - rho^2)
    alpha <- C * Pi
    z <- rdirichlet(n, alpha)

    Y <- matrix(NA, k, n)
    for (i in 1:n) {
        Y[,i] <- rmultinom(1, prob = z[i,], size = m[i])
    }

    return(Y)
}

d.dm <- function(x, Pi, rho, m, log = FALSE)
{
    k <- length(Pi)
    n <- ncol(x)
    if (length(m) == 1) m <- rep(m, n)
    C <- rho^(-2) * (1 - rho^2)
    alpha <- C * Pi

    log.part <- matrix(NA, n, k)
    for (j in 1:k) {
        log.part[,j] <- lgamma(alpha[j] + x[j,]) - lgamma(alpha[j]) - lgamma(x[j,] + 1)
    }

    log.ff <- lgamma(m+1) + lgamma(C) - lgamma(C + m) + rowSums(log.part)
    if (log) return(log.ff)
    else return(exp(log.ff))
}
```

As in the function `d.multmix`, we evaluate the DM density in a vectorized manner over n observations. Now we carry out our simple test to ensure `d.dm` and `r.dm` are implemeting the same distribution.

```r
n <- 1000000
m <- 20
Pi <- c(1/6, 2/6, 3/6)
rho <- 1/4
y <- r.dm(n, Pi, rho, m)

for (i in seq(1,m+1)) {
    for (j in seq(1,m+1)) {
        x <- matrix( c(i-1, j-1, m - (i-1) - (j-1)), 3, 1 )
        if (x[3,] > m || x[3,] < 0) next
        cat("x =", x)
        printf("\tdensity = %g, ", d.dm(x, Pi, rho, m))
        printf("prop = %g\n", sum(y[1,] == x[1,] & y[2,] == x[2,]) / n)
    }
}
```

```
x = 0 0 20    density = 0.000329156, prop = 0.000332
x = 0 1 19    density = 0.0012421, prop = 0.001232
x = 0 2 18    density = 0.00277645, prop = 0.002772
x = 0 3 17    density = 0.00475963, prop = 0.004693
x = 0 4 16    density = 0.00688628, prop = 0.006862
...
```

2.9 Random-Clumped Multinomial

Random-clumped multinomial (RCM) is the multinomial analogue of random-clumped binomial. See Section 7.8 of OMSAS. RCM arises by considering variables

$$\begin{aligned}\boldsymbol{Y} &\sim \mathsf{Mult}(1, \boldsymbol{\pi}),\\ \boldsymbol{X} &\sim \mathsf{Bin}(m - N, \boldsymbol{\pi}),\\ N &\sim \mathsf{Bin}(m, \rho).\end{aligned}$$

The observation $\boldsymbol{T} = N\boldsymbol{Y} + \boldsymbol{X}$ follows the RCB distribution with parameters $\boldsymbol{\pi}$ and ρ, notated as $T \sim \mathsf{RCM}_k(m, \boldsymbol{\pi}, \rho)$. As in RCB, $\boldsymbol{Y}$ represents selection of a leader, N is the number of trials that follows the leader, and $\boldsymbol{X}$ represents the remaining trials that are selected independently. Therefore, $\boldsymbol{T}$ is a sum of multinomial trials which are dependent, and the degree of depedence is increased as $\rho \uparrow 1$. We may interpret $\boldsymbol{\pi}$ as the category probabilities for the RCM trials, and $\rho \in (0, 1)$ as the probability of following the leader. The density of RCM can be expressed as a finite mixture of k multinomial densities, with mixing proportions $\boldsymbol{\pi}$ and the jth mixture component as $\mathsf{Mult}_k(m, \boldsymbol{p}_j)$, where

$$\boldsymbol{p}_j = (1 - \rho)\boldsymbol{\pi} + \rho \boldsymbol{e}_j, \quad j = 1, \ldots, k$$

and $\boldsymbol{e}_j$ is the jth column of the $k \times k$ identity matrix. The expectation and variance of $\boldsymbol{T}$ turn out to be identical to the expressions from Dirichlet-multinomial, namely

$$\mathrm{E}(\boldsymbol{T}) = m\boldsymbol{\pi}, \quad \mathsf{Var}(\boldsymbol{T}) = m[1 + \rho^2(m - 1)] \left[\mathsf{Diag}(\boldsymbol{\pi}) - \boldsymbol{\pi}\boldsymbol{\pi}^T\right].$$

The following code evaluates and draws from the density in R.

File: /home/araim/R/rcm.R

```r
r.rcm <- function(n, Pi, rho, m)
{
    k <- length(Pi)
    stopifnot( 0 < rho && rho < 1 )
    if (length(m) == 1) m <- rep(m, n)
    Y <- rmultinom(n, prob = Pi, size = 1)
    N <- rbinom(n, prob = rho, size = m)
    X <- sapply(m-N, rmultinom, n = 1, prob = Pi)
    N.mat <- matrix(N, k, n, byrow = TRUE)
    N.mat*Y + X
}

d.rcm <- function(x, Pi, rho, m, log = FALSE)
{
    k <- length(Pi)
    Pi <- normalize(Pi) # We get strange results if Pi isn't normalized
    P <- (1 - rho) * Pi + diag(rho, k, k)
    d.multmix(x, P, Pi, m, log)
}
```

Notice that in `d.rcm`, we make use of the `d.multmix` function presented in Section 2.7. Here is our test to ensure `d.rcm` and `r.rcm` are implemeting the same distribution.

```
n <- 1000000
m <- 20
Pi <- c(1/6, 2/6, 3/6)
rho <- 1/4
y <- r.rcm(n, Pi, rho, m)

for (i in seq(1,m+1)) {
    for (j in seq(1,m+1)) {
        x <- matrix( c(i-1, j-1, m - (i-1) - (j-1)), 3, 1 )
        if (x[3,] > m || x[3,] < 0) next
        cat("x =", x)
        printf("\tdensity = %g, ", d.rcm(x, Pi, rho, m))
        printf("prop = %g\n", sum(y[1,] == x[1,] & y[2,] == x[2,]) / n)
    }
}
```

```
x = 0 0 20    density = 4.13605e-05, prop = 3.2e-05
x = 0 1 19    density = 0.000330906, prop = 0.000317
x = 0 2 18    density = 0.0012577, prop = 0.001253
x = 0 3 17    density = 0.00302045, prop = 0.002988
x = 0 4 16    density = 0.00514576, prop = 0.005088
...
```

3 Numerical Framework for Maximum Likelihood

In this section, we develop a framework for numerical maximum likelihood which can be used to replicate analysis of examples in OMSAS, which have been carried out in `SAS` via `PROC NLMIXED` and `PROC GLIMMIX`. These examples make use of BB, RCB, and other distributions which are not readily fit in `R` without some work (i.e. programming or searching for an appropriate package). Given a likelihood model, data, and analysis objectives, we would like to obtain estimates, confidence intervals, and model fit statistics via MLE without too much programming for each new problem.

The reader may want to skip to Section 6 to see the example data analyses first, then return here to see the underlying framework. A complete version of the code is provided as the package `OverdispersionModelsInR`; see Section 1.1.

3.1 Methodology

Suppose $Y_i \overset{\text{ind}}{\sim} f_i(y \mid \boldsymbol{\theta})$ for $i = 1, \ldots, n$ so that we observe $y_1, \ldots, y_n$. Often, $f_i(y \mid \boldsymbol{\theta}) = f(y \mid \boldsymbol{\theta}, \boldsymbol{x}_i)$ so that the f_i vary only by a covariate $\boldsymbol{x}_i$. Generically, we will denote all the data needed for the given likelihood as $\mathcal{D}$. The log-likelihood is $\log L(\boldsymbol{\theta}) = \sum_{i=1}^n \log[f_i(y_i \mid \boldsymbol{\theta})]$, and the maximum likelihood estimator (MLE) $\hat{\boldsymbol{\theta}}$ is found by maximizing $\log L(\boldsymbol{\theta})$ over $\boldsymbol{\theta}$ in the parameter space Θ. Numerical optimization routines may be used when the maximization cannot worked out in closed form. Unconstrained optimization routines are the most commonly available in software, but the space Θ often has natural constraints. For example, there may be range restrictions on individual parameters (e.g. the probability of success of a binomial must be between 0 and 1), or the restrictions may involve multiple parameters (e.g. the probabilities of a multinomial must be non-negative and sum to 1). For our optimization routine, we assume that the Hessian

$$H(\hat{\boldsymbol{\theta}}) = \frac{\partial^2}{\partial \boldsymbol{\theta} \partial \boldsymbol{\theta}^T} \log L(\boldsymbol{\theta}) \bigg|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}}$$

will be returned along with the solution $\hat{\boldsymbol{\theta}}$.

One way to compute $\hat{\boldsymbol{\theta}}$ using an unconstrained optimizer is to consider $\boldsymbol{\theta} = \boldsymbol{\theta}(\boldsymbol{\phi})$ as a smooth transformation of unconstrained variables $\boldsymbol{\phi} \in \mathbb{R}^q$. We will assume that the transformation is invertible. Then maximizing $\log L(\boldsymbol{\theta}(\boldsymbol{\phi}))$ over $\boldsymbol{\phi}$ yields a solution $\hat{\boldsymbol{\phi}}$ and Hessian $H(\hat{\boldsymbol{\phi}})$. Multivariate optimization routines often require a starting value $\boldsymbol{\phi}^{(0)}$ to be provided as an input, and we will assume this in our framework.

We may also be interested in additional functions $\boldsymbol{\xi}(\boldsymbol{\theta})$ of $\boldsymbol{\theta}$ which do not appear in the likelihood. For example, in a binomial analysis of a 2×2 table, we are often interested in the odds-ratio for a treatment vs. control. The $\boldsymbol{\xi}$ of interest varies from problem to problem, even when the same model is used. We define $\boldsymbol{\psi}(\boldsymbol{\phi}) = (\boldsymbol{\theta}, \boldsymbol{\xi})$ to be the combined parameters of interest: those needed to compute the likelihood plus additional parameters of interest.

Given the solution $\hat{\boldsymbol{\phi}}$ and Hessian $H(\hat{\boldsymbol{\phi}})$, $\hat{\boldsymbol{\psi}} = \boldsymbol{\psi}(\hat{\boldsymbol{\phi}})$ is an MLE of $\boldsymbol{\psi}$ by the invariance property. The large sample variance $\mathsf{V}(\hat{\boldsymbol{\psi}})$ can also be estimated in the following way. If the large sample distribution of $\hat{\boldsymbol{\phi}}$ is $\mathsf{N}(\boldsymbol{\phi}, \mathcal{I}^{-1}(\boldsymbol{\phi}))$ with $\mathcal{I}(\boldsymbol{\phi})$ denoting the Fisher information matrix, then $\widehat{\mathsf{V}}(\hat{\boldsymbol{\phi}}) = [-H(\hat{\boldsymbol{\phi}})]^{-1}$ is an estimate of $\mathsf{V}(\hat{\boldsymbol{\phi}})$. The large sample distribution of $\hat{\boldsymbol{\psi}}$ is then

$$\mathsf{N}\left[\boldsymbol{\psi}, \left(\frac{\partial \boldsymbol{\psi}}{\partial \boldsymbol{\phi}}\right) \mathcal{I}^{-1}(\boldsymbol{\phi}) \left(\frac{\partial \boldsymbol{\psi}}{\partial \boldsymbol{\phi}}\right)^T\right],$$

whose variance can be estimated by

$$\widehat{\mathsf{V}}(\hat{\boldsymbol{\psi}}) = \left(\frac{\partial \boldsymbol{\psi}}{\partial \boldsymbol{\phi}}\right) [-H(\boldsymbol{\phi})]^{-1} \left(\frac{\partial \boldsymbol{\psi}}{\partial \boldsymbol{\phi}}\right)^T \Bigg|_{\boldsymbol{\phi}=\hat{\boldsymbol{\phi}}}.$$

The use of a numerical differentiation routine may be used to compute $\partial \boldsymbol{\psi} / \partial \boldsymbol{\phi}$ at a given $\hat{\boldsymbol{\phi}}$. Using this method, it is possible to compute $\hat{\boldsymbol{\psi}}$ and $\widehat{\mathsf{V}}(\hat{\boldsymbol{\psi}})$ numerically, given a likelihood, without additional derivations by hand. Furthermore, standard errors $\mathrm{SE}(\hat{\psi}_j)$ for $j = 1, \ldots, \dim(\boldsymbol{\psi})$ may be computed from the diagonal of $\widehat{\mathsf{V}}(\hat{\boldsymbol{\psi}})$. Corresponding t-values may be computed as t-value$_j = \hat{\psi}_j / \mathrm{SE}(\hat{\psi}_j)$, and p-values for the tests $H_0 : \psi_j = 0$ vs. $H_1 : \psi_j \neq 0$ may be computed using p-value$_j = \mathrm{P}(|t| \geq$ t-value$_j)$, where t is distributed from Student's t-distribution with n degrees of freedom. We may compute components of the gradient $\frac{\partial}{\partial \psi} \log L(\boldsymbol{\theta})$ at the solution $\hat{\boldsymbol{\theta}}$, which is a useful diagnostic provided by `PROC NLMIXED`.

To summarize, the inputs to our numerical MLE framework will be:

- The data $\mathcal{D}$ needed to evaluate the likelihood, including any observations and covariates.
- The likelihood $L(\boldsymbol{\theta})$ describing our statistical model.
- Unconstrained start values $\boldsymbol{\phi}^{(0)}$ for the optimization.
- A transformation $\boldsymbol{\theta}(\boldsymbol{\phi}) \in \Theta$ to obtain valid likelihood parameters from a given $\boldsymbol{\phi}$.
- Additional functions $\boldsymbol{\xi}(\boldsymbol{\theta})$ of $\boldsymbol{\theta}$ which are of interest to us.

The outputs will be:

- Estimates $\hat{\boldsymbol{\psi}}$.
- Estimate of the variance $\widehat{\mathrm{V}}(\hat{\boldsymbol{\psi}})$.
- Standard errors $\mathrm{SE}(\hat{\psi}_j)$.
- t-value$_j = \hat{\psi}_j / \mathrm{SE}(\hat{\psi}_j)$.
- p-value$_j = \mathrm{P}(|t| \geq$ t-value$_j)$ where $t \sim \mathrm{t}_n$.
- The gradient $\frac{\partial}{\partial \psi_j} \log L(\boldsymbol{\theta})$ evaluated at $\boldsymbol{\theta} = \hat{\boldsymbol{\theta}}$.
- Log-likelihood, AIC, BIC, and other functions of $\log L(\hat{\boldsymbol{\theta}})$ that are generally useful.

3.2 R Code

We will use two numerical tools in R to program our MLE framework: the optim function for optimization and the numderiv package for numerical derivatives. To demonstrate optim, consider maximizing the function $f(\boldsymbol{x}) = -\sum_{i=1}^n x_i^2$ over $\boldsymbol{x} \in \mathbb{R}^n$. In this simple example, it is immediate that $\boldsymbol{x}^* = \mathbf{0}$ maximizes f with $f(\boldsymbol{x}^*) = 0$. Let us do this computation in R.

```
> f <- function(x) { -sum(x^2) }
> x0 <- rep(1/2, 10)
> res <- optim(x0, f, method = "L-BFGS-B", control = list(fnscale = -1))
> res
$par
 [1] 2.414044e-20 2.414044e-20 2.414044e-20 2.414044e-20 2.414044e-20 2.414044e-20
 [7] 2.414044e-20 2.414044e-20 2.414044e-20 2.414044e-20

$value
[1] -5.827608e-39

$counts
function gradient
       4        4

$convergence
[1] 0

$message
[1] "CONVERGENCE: NORM OF PROJECTED GRADIENT <= PGTOL"
```

Notice that the dimension of x is not specified in f, and is only determined by the initial value x0 used in optim. We have selected the Quasi-Newton method L-BFGS-B as our optimization algorithm, and have specified fnscale = -1 to tell optim that this is a maximization problem rather than a minimization problem. We obtain a solution that is not exactly $\boldsymbol{x}^* = \mathbf{0}$, but is very close. The value 0 for res$convergence indicates that the algorithm has converged. We could also pass the argument hessian = TRUE to request that optim return the Hessian matrix.

To install the package numDeriv from CRAN, use the following command.

```
> install.packages("numDeriv")
```

Consider computing the Jacobian of the function $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^n$ defined by $\boldsymbol{f}(\boldsymbol{x}) = (x_1, x_1 + x_2, \ldots, x_1 + \cdots + x_n)$. In this simple example, the Jacobian is computed exactly as the lower-triangular matrix of ones

$$J\{\boldsymbol{f}\}(\boldsymbol{x}) = \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 \\ 1 & 1 & \cdots & 0 & 0 \\ 1 & 1 & \ddots & 1 & 0 \\ 1 & 1 & \cdots & 1 & 1 \end{pmatrix}.$$

Let us carry out the computation in R. Notice that we must specify a point x0 at which to evaluate the Jacobian, even though $J\{\boldsymbol{f}\}(\boldsymbol{x})$ is constant over $\boldsymbol{x}$ in this particular example. Also note that the function $\boldsymbol{f}$ is implemented in R as the cumsum function.

```
> library(numDeriv)
> x0 <- rep(0, 5)
> jacobian(cumsum, x0)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    0    0    0    0
[2,]    1    1    0    0    0
[3,]    1    1    1    0    0
[4,]    1    1    1    1    0
[5,]    1    1    1    1    1
```

Let us proceed in constructing the MLE framework using the RCB distribution as an example. Recall that $\boldsymbol{\phi}$ are the unconstrained optimization variables and $\boldsymbol{\theta}(\boldsymbol{\phi})$ are the variables needed to evaluate the likelihood. For the setting $Y_i \overset{\text{iid}}{\sim}$ RCB(π, ρ), we will take $\boldsymbol{\phi} \in \mathbb{R}^2$ and let $\boldsymbol{\theta} = (\pi, \rho)$ so that $\pi = G(\phi_1)$ and $\rho = G(\phi_2)$ where G is the logistic CDF. This will ensure that π and ρ are both constained to the interval $(0, 1)$. We can construct the log-likelihood in the following way.

File: /home/araim/R/likelihood-rcb.R

```
source("/home/araim/R/rcb.R")

theta.tx <- function(phi)
{
    list(Pi = plogis(phi[1]), rho = plogis(phi[2]))
}

loglik <- function(phi, Data)
{
    theta <- theta.tx(phi)
    sum( d.rcb(Data$y, Pi = theta$Pi, rho = theta$rho,
        m = Data$m, log = TRUE) )
}
```

The function theta.tx is the transformation from $\boldsymbol{\phi}$ to $\boldsymbol{\theta}$. Here we have returned a list for convenience, so that later we can use an expression like theta$rho to access ρ rather than theta[2] (which is what we would use if we returned a numerical vector). We also require that the likelihood data $\mathcal{D}$ be made available as the list Data. Now we can evaluate the likelihood given a particular dataset, say, one we have drawn ourselves.

```
> y <- r.rcb(n = 100, Pi = 0.25, rho = 0.4, m = 20)
> Data <- list(y = y, m = 20)
> phi <- c(0,0)
> loglik(phi, Data)
[1] -334.9084
```

We now compute the MLE of $\boldsymbol{\phi}$, although the optimization variables themselves may not be our eventual interest.

```
> phi.init <- c(0,0)
> res <- optim(phi.init, loglik, method = "L-BFGS-B", control = list(fnscale = -1),
+ hessian = TRUE, Data = Data)
> phi.hat <- res$par
> V.phi <- solve(-res$hessian)
> print(phi.hat)
[1] -1.1014219 -0.5168278
> print(V.phi)
            [,1]        [,2]
[1,] 0.006599719 0.001625827
[2,] 0.001625827 0.009781809
```

Next we will transform to the estimates and estimated variances for $\boldsymbol{\psi} = (\boldsymbol{\theta}, \boldsymbol{\xi})$. Suppose we are interested in the additional parameters $\boldsymbol{\xi} = (\log \pi, \rho^2)$ as well.

```
File: /home/araim/R/transform-rcb.R
extra.tx <- function(theta)
{
    list(log.Pi = log(theta$Pi), rho.sq = theta$rho^2)
}

psi.tx <- function(phi)
{
    theta <- theta.tx(phi)
    psi1 <- unlist(theta)
    psi2 <- unlist(extra.tx(theta))
    psi <- c(psi1, psi2)
    return(psi)
}

theta.hat <- theta.tx(phi.hat)
xi.hat <- extra.tx(theta.hat)
psi.hat <- psi.tx(phi.hat)

V.phi <- solve(-res$hessian)
J.tx <- jacobian(psi.tx, phi.hat)
V.psi <- J.tx %*% V.phi %*% t(J.tx)
```

Notice that `psi.tx` returns a `vector` rather than a `list`. This makes it usable with the `jacobian` function. We now have the following results.

```
> unlist(theta.hat)
       Pi       rho
0.2494736 0.3735943
> unlist(xi.hat)
    log.Pi     rho.sq
-1.3884023  0.1395727
> psi.hat
        Pi        rho     log.Pi     rho.sq
 0.2494736  0.3735943 -1.3884023  0.1395727
> V.phi
            [,1]        [,2]
[1,] 0.006599719 0.001625827
[2,] 0.001625827 0.009781809
> J.tx
          [,1]      [,2]
[1,] 0.1872365 0.0000000
[2,] 0.0000000 0.2340216
[3,] 0.7505264 0.0000000
[4,] 0.0000000 0.1748583
> V.psi
             [,1]         [,2]         [,3]         [,4]
[1,] 2.313697e-04 0.0000712395 0.0009274318 5.322934e-05
[2,] 7.123950e-05 0.0005357116 0.0002855593 4.002776e-04
[3,] 9.274318e-04 0.0002855593 0.0037175555 2.133667e-04
[4,] 5.322934e-05 0.0004002776 0.0002133667 2.990828e-04
```

The `unlist` function converts a `list` into a `vector`; here we have used it to display the output more compactly for our `list` variables. Now `psi.hat` represents $\hat{\boldsymbol{\psi}}$ and `V.psi` represents $\widehat{\mathsf{V}}(\hat{\boldsymbol{\psi}})$. We can now compute a table with the estimates, standard errors, t-values, and p-values.

```
> df <- n
> se <- sqrt(diag(V.psi))
> t.val <- psi.hat / se
> p.val <- 2 * (1 - pt(abs(t.val), df))
> gr <- J.tx %*% grad(loglik, x = phi.hat, Data = Data)
> estimates <- cbind(psi.hat, se, t.val, p.val, gr)
> colnames(estimates) <- c("Estimate", "SE", "t-val", "P(|t|>t-val)", "Gradient")
> estimates
         Estimate         SE      t-val P(|t|>t-val)     Gradient
Pi      0.2580914 0.01620894  15.922784            0 3.363525e-06
rho     0.4404795 0.02613963  16.851023            0 3.855720e-06
log.Pi -1.3544414 0.06280309 -21.566479            0 1.303230e-05
rho.sq  0.1940222 0.02302794   8.425511            0 3.396731e-06
```

Note that `grad` is a function in the `numDeriv` package which computes the gradient of a function `f` evaluated at a point `x`. We may also be interested in quantites such as $\log L(\hat{\boldsymbol{\theta}})$,

$$\begin{aligned}
\mathsf{AIC} &= -2\log L(\hat{\boldsymbol{\theta}}) + 2q, \\
\mathsf{BIC} &= -2\log L(\hat{\boldsymbol{\theta}}) + q\log(n), \\
\mathsf{AICC} &= -2\log L(\hat{\boldsymbol{\theta}}) + \frac{2nq}{n-q-1}.
\end{aligned}$$

where $q = \dim(\boldsymbol{\theta})$. Note that the variable name q is reserved for the "quit" function in R, so we use qq instead.

```
> qq <- length(unlist(theta.hat))
> n <- length(Data$y)
> ( loglik.hat <- res$value )
[1] -257.6272
> ( aic <- -2 * loglik.hat + 2*qq )
[1] 519.2543
> ( aicc <- -2 * loglik.hat + 2*qq*n / (n-qq-1) )
[1] 519.378
> ( bic <- -2 * loglik.hat + qq*log(n) )
[1] 524.4647
```

Listing 1 presents code for the function `fit-mle.R`. It computes the table of estimates and some additional statistics based on numerical maximization of the log-likelihood given ingredients: a `loglik` function, a `theta.tx` transformation to compute likelihood parameters, an `extra.tx` transformation to compute additional parameters of interest, an initial value `phi.init` for the optimizer, and the `Data`.

File: /home/araim/R/fit-mle.R

```
library(numDeriv)

fit.mle <- function(phi.init, loglik, theta.tx, extra.tx, Data)
{
    stopifnot(!is.null(Data$n))
    n <- Data$n

    psi.tx <- function(phi)
    {
        theta <- theta.tx(phi)
        psi1 <- unlist(theta)
        psi2 <- unlist(extra.tx(theta))
        psi <- c(psi1, psi2)
        return(psi)
    }

    optim.res <- optim(par = phi.init, fn = loglik, method = "L-BFGS-B",
        control = list(fnscale = -1), hessian = TRUE, Data = Data)

    phi.hat <- optim.res$par
    theta.hat <- theta.tx(phi.hat)
    xi.hat <- extra.tx(theta.hat)
    psi.hat <- psi.tx(phi.hat)

    V.phi <- solve(-optim.res$hessian)
    J.tx <- jacobian(psi.tx, phi.hat)
    V.psi <- J.tx %*% V.phi %*% t(J.tx)

    loglik.hat <- optim.res$value
    qq <- length(unlist(theta.hat))
    aic <- -2 * loglik.hat + 2*qq
    aicc <- -2 * loglik.hat + 2*qq*n / (n-qq-1)
    bic <- -2 * loglik.hat + qq*log(n)

    df <- n
    se <- sqrt(diag(V.psi))
    t.val <- psi.hat / se
    p.val <- 2 * (1 - pt(abs(t.val), df = df))
    gr <- J.tx %*% grad(loglik, x = phi.hat, Data = Data)

    estimates <- cbind(psi.hat, se, t.val, p.val, gr)
    colnames(estimates) <- c("Estimate", "SE", "t-val", "P(|t|>t-val)", "Gradient")

    res <- list(estimates = estimates, loglik = loglik.hat, aic = aic,
        aicc = aicc, bic = bic, vcov = V.psi, optim.res = optim.res)
    class(res) <- "mle.fit"
    return(res)
}
```

Listing 1: Code for main function of numerical MLE framework.

Notice that before the final result is returned, we have specified that is of class `mle.fit`, which will be used later. Also, we require `Data` to contain a sample size `n` which is needed for calculation of AIC, BIC, etc. Next we present a function `fit-rcb-mle.R` which fits the likelihood of $Y_i \overset{\text{iid}}{\sim} \text{RCB}(\pi, \rho)$ using the `fit.mle` function.

File: /home/araim/R/fit-rcb-mle.R
```
source("rcb.R")
source("fit-mle.R")

fit.rcb.mle <- function(y, m, extra.tx)
{
    Data <- list(y = y, m = m, n = length(y))
    qq <- 2
    phi.init <- rep(0, qq)

    theta.tx <- function(phi)
    {
        list(Pi = plogis(phi[1]), rho = plogis(phi[2]))
    }

    loglik <- function(phi, Data)
    {
        theta <- theta.tx(phi)
        sum( d.rcb(Data$y, Pi = theta$Pi, rho = theta$rho,
            m = Data$m, log = TRUE) )
    }

    fit.out <- fit.mle(phi.init, loglik, theta.tx, extra.tx, Data)
    fit.out$description <- "y[i] ~iid~ RCB(Pi, rho)"
    return(fit.out)
}
```

Notice that the only inputs are the data `y` and `m`, and the additional transformation function `extra.tx`. Before returning the fit, we add a descriptive string that specifies the model. Fitting the RCB likelihood can now be done in a few lines.

```
> source("fit-rcb-mle.R")
> n <- 100
> m <- 20
> y <- r.rcb(n, Pi = 1/4, rho = 0.4, m = m)
> extra.tx <- function(theta) { list(log.Pi = log(theta$Pi), rho.sq = theta$rho^2) }
> fit.out <- fit.rcb.mle(y, m, extra.tx)
> fit.out$estimates
         Estimate         SE      t-val P(|t|>t-val)     Gradient
Pi      0.2580914 0.01620894  15.922784            0 3.363525e-06
rho     0.4404795 0.02613963  16.851023            0 3.855720e-06
log.Pi -1.3544414 0.06280309 -21.566479            0 1.303230e-05
rho.sq  0.1940222 0.02302794   8.425511            0 3.396731e-06
```

The `list` `fit.out` contains many fields, and printing it gives a long output. We can format the output to get a nicer display by adding the function `print.fit.out` to the file `fit-mle.R`. Let us also add a function `printf` for printing of formatted messages, similar to `printf` in `C`. We use the built-in `R` function `sprintf` to handle formatting.

```
printf <- function(msg, ...) { cat(sprintf(msg, ...)) }

print.mle.fit <- function(fit.out)
{
    printf("Fit for model:\n")
    printf("%s\n", fit.out$description)
    printf("--\n")

    estimates <- fit.out$estimates
    estimates[,1] <- round(estimates[,1], 4)
    estimates[,2] <- round(estimates[,2], 4)
    estimates[,3] <- round(estimates[,3], 4)

    print(estimates)
    printf("--\n")
    printf("LogLik = %0.4f\n", fit.out$loglik)
    printf("AIC = %0.4f\n", fit.out$aic)
    printf("AICC = %0.4f\n", fit.out$aicc)
    printf("BIC = %0.4f\n", fit.out$bic)
}
```

```
> fit.out
Fit for model:
y[i] ~iid~ RCB(m[i], Pi, rho)
--
         Estimate         SE      t-val P(|t|>t-val)     Gradient
Pi      0.2580914 0.01620894  15.922784            0 3.363525e-06
rho     0.4404795 0.02613963  16.851023            0 3.855720e-06
log.Pi -1.3544414 0.06280309 -21.566479            0 1.303230e-05
rho.sq  0.1940222 0.02302794   8.425511            0 3.396731e-06
--
LogLik = -243.8098
AIC = 491.6195
AICC = 491.7433
BIC = 496.8299
```

It is not a coincidence that we named our function `print.mle.fit`, and that the object returned from `fit.mle` has been given class type `mle.fit`. This is a simple use of object oriented programming in `R`. When we call `print` on an object of type `fit.mle`, `R` will see that the function `print.mle.fit` exists, and will use it to print our object. This is because the `print` function has been defined specially. Let us check the definition of the `print` function.

```
> print
function (x, ...)
UseMethod("print")
<bytecode: 0x2b8b258>
<environment: namespace:base>
```

The declaration `UseMethod("print")` lets us know that `R` will try to find a specialized `print` function for our object if we simply write `print(object)`. Also notice that when we viewed our results, we simply wrote `fit.out`, which is a shorthand for `print(fit.out)`.

4 Scoring Framework for Maximum Likelihood

It may be desirable to avoid use of sophisticated optimization routines in statistical applications. Optimization based on Newton-Raphson or scoring uses simpler iterations, which are more easily coded and often work very well. In this section we will briefly present a framework for scoring type algorihms.

4.1 Methodology

Denote $\boldsymbol{\theta}$ generically as the parameter of interest, $\boldsymbol{\theta}^{(g)}$ as the gth iterate in an iterative algorithm, $S(\boldsymbol{\theta})$ as the score function, $\boldsymbol{H}(\boldsymbol{\theta})$ as the Hessian of the score function, and $\mathcal{I}(\boldsymbol{\theta})$ as the Fisher information matrix (FIM). Two widely used algorithms are Newton-Raphson, with iterations

$$\boldsymbol{\theta}^{(g+1)} = \boldsymbol{\theta}^{(g)} - \left\{\boldsymbol{H}(\boldsymbol{\theta}^{(g)})\right\}^{-1} S(\boldsymbol{\theta}^{(g)}), \quad g = 0, 1, 2, \dots \tag{1}$$

and Fisher scoring

$$\boldsymbol{\theta}^{(g+1)} = \boldsymbol{\theta}^{(g)} + \left\{\mathcal{I}(\boldsymbol{\theta}^{(g)})\right\}^{-1} S(\boldsymbol{\theta}^{(g)}), \quad g = 0, 1, 2, \dots \tag{2}$$

which replaces the Hessian with the information matrix. These algorithms have a fast rate of convergence to a solution (of the likelihood equations) when given a starting value $\boldsymbol{\theta}^{(0)}$ near the solution. However, they may have difficulty making progress if started from an arbitrary starting value. Raim et al. (2014) find this to be the case when fitting multinomial finite mixtures, whose likelihood may be relatively difficult to maximize numerically compared to, say, the standard multinomial, DM, or RCM. Raim et al. (2014) consider an approximate scoring algorithm,

$$\boldsymbol{\theta}^{(g+1)} = \boldsymbol{\theta}^{(g)} + \left\{\widetilde{\mathcal{I}}(\boldsymbol{\theta}^{(g)})\right\}^{-1} S(\boldsymbol{\theta}^{(g)}), \quad g = 0, 1, 2, \dots, \tag{3}$$

which is more robust to the starting value but slower than Fisher scoring, and uses an approximation $\widetilde{\mathcal{I}}(\boldsymbol{\theta})$ to the information matrix. The matrix $\widetilde{\mathcal{I}}(\boldsymbol{\theta})$ is the complete data information matrix of the observed $\boldsymbol{Y}$ and the latent mixing process Z. In the case of a finite mixture, Z is taken to be the latent subpopulation indicator. The matrix $\widetilde{\mathcal{I}}(\boldsymbol{\theta})$ often has a form which can be derived analytically, even when $\mathcal{I}(\boldsymbol{\theta})$ does not, which is one advantage of approximate scoring. The convergence rate of approximate scoring is seen to be very similar to that of Expectation-Maximization. Raim et al. (2014) suggest a hybrid scoring method, using approximate scoring iterations until some initial convergence criteria, then proceeding with Fisher scoring until the desired convergence. This provides a fast and robust algorithm. A hybrid of approximate scoring and Newton-Raphson may be used instead when the FIM can not be computed, and the Hessian of the log-likelihood can at least be computed numerically.

4.2 R Code

Notice that all algorihms discussed above — Newton-Raphson, Fisher scoring, approximate scoring, and the two hybrid methods — are essentially of the same form, swapping out the matrix used as the Hessian. To implement the algorithms, the code for Newton-Raphson is given below in the `fit.nr` function.

File: /home/araim/R/scoring-nr.R

```r
fit.nr <- function(theta.init, loglik, score = NULL, hess = NULL,
    Data, max.iter = Inf, tol = 1e-6)
{
    ll <- -Inf
    iter <- 0
    delta <- Inf
    theta <- theta.init

    if (is.null(score)) {
        score <- function(theta, Data) { grad(func = loglik, x = theta, Data = Data,
            method.args = list(d = 1e-4)) }
    }

    if (is.null(hess)) {
        hess <- function(theta, Data) { hessian(func = loglik, x = theta, Data = Data,
            method.args = list(d = 1e-4)) }
    }

    while (iter < max.iter && abs(delta) > tol) {
        iter <- iter + 1
        S <- score(theta, Data)
        H <- hess(theta, Data)
        theta <- theta - solve(H, S)

        ll.old <- ll
        ll <- loglik(theta, Data)
        delta <- ll - ll.old
    }

    list(loglik = ll, iter = iter, tol = delta, converged = (iter < max.iter), theta.hat = theta)
}
```

As a convenience to the user, we compute the score or Hessian numerically if they are not supplied. The other algorithms can now be implemented simply by calling the `fit.nr` function with different arguments for `hess`.

File: /home/araim/R/scoring-rest.R

```r
fit.fs <- function(theta.init, loglik, score, fim, Data, max.iter = Inf,
    tol = 1e-6)
{
    hess <- function(theta, Data) { -fim(theta, Data) }
    fit.nr(theta.init, loglik, score, hess, Data, max.iter, tol)
}

fit.afs <- function(theta.init, loglik, score, afim, Data, max.iter = Inf,
    tol = 1e-6)
{
    hess <- function(theta, Data) { -afim(theta, Data) }
    fit.nr(theta.init, loglik, score, hess, Data, max.iter, tol)
}

fit.fs.hybrid <- function(theta.init, loglik, score, fim, afim, Data,
    max.iter = Inf, tol = 1e-6, warmup.tol = 1e-4)
{
    out.afs <- fit.afs(theta.init, loglik, score, afim, Data, max.iter, tol = warmup.tol)
    fit.fs(out.afs$theta.hat, loglik, score, fim, Data, max.iter - out.afs$iter, tol)
}

fit.nr.hybrid <- function(theta.init, loglik, score, hess, afim, Data,
    max.iter = Inf, tol = 1e-6, warmup.tol = 1e-4)
{
    out.afs <- fit.afs(theta.init, loglik, score, afim, Data, max.iter, tol = warmup.tol)
    fit.nr(out.afs$theta.hat, loglik, score, hess, Data, max.iter - out.afs$iter, tol)
}
```

To make the code usable, ingredients such as the log-likelihood, score, FIM, and approximate FIM must be provided for each model under consideration. The reader may refer to the source code for the OverdispersionModelsInR package, which implements the needed functions for RCM and DM iid models. The functions for each likelihood model are grouped into a "family", and abbreviated calls to the algorithms are also possible as follows.

```
dm <- dm.scoring(k = 4)
theta.init <- c(1/4,1/4,1/4, 0.5)
out.nr <- fit.family.afs(theta.init, dm, Data)
```

In this example, we create an object representing the DM family with $k = 4$ categories. The function `fit.family.afs` extracts the log-likelihood, score, and approximate FIM, and calls the original `fit.afs` function.

As in the numerical MLE framework discussed in Section 3, we can add support to estimate extra quantities of interest, compute the table of estimates and standard errors, and print a nice display of the results. The reader may refer to the source code for the OverdispersionModelsInR package to see these additions. It is also possible to start with real-valued optimization variables $\boldsymbol{\phi}$ and provide a transformation to likelihood parameters $\boldsymbol{\theta}$, which may help the algorithms to avoid wandering outside the parameter space (which essentially dooms them to failure), but this has currently not been implemented.

5 Goodness of Fit Test for Binomial Data

This section presents a goodness-of-fit test for binomial data, and gives R code to implement it. The reader may want to jump ahead to Section 6.4 to see the procedure applied to an example, then return to this section to look through the details.

5.1 Methodology

To compare several variations of the binomial model on the same dataset, consider the goodness-of-fit (GOF) test

$$H_0 : Y_i \overset{\text{ind}}{\sim} f_i(t_i \mid m_i, \boldsymbol{\theta}) \text{ for some } \boldsymbol{\theta} \in \Theta \quad \text{vs.} \quad H_1 : \text{Not},$$

where f_i is fully specified up to parameter $\boldsymbol{\theta}$ (which is usually unknown in practice) in the space $\Theta \subseteq \mathbb{R}^q$. For binomial-type data with m_i varying with observations, Neerchal and Morel (1998) proposed the following variation to the usual Pearson chi-square test statistic (see also OMSAS section 5.3). Suppose $\mathcal{A}_1, \ldots, \mathcal{A}_r$ are disjoint intervals that cover $[0, 1]$, and define the GOF test statistic

$$X(\boldsymbol{\theta}) = \sum_{\ell=1}^{r} \frac{[O_\ell - E_\ell(\boldsymbol{\theta})]^2}{E_\ell(\boldsymbol{\theta})}, \tag{4}$$

where

$$E_\ell(\boldsymbol{\theta}) = \sum_{i=1}^{n} \sum_{t=0}^{m_i} \mathrm{P}(t \mid m_i, \boldsymbol{\theta}) I\left(\frac{t}{m_i} \in \mathcal{A}_\ell\right) \quad \text{and} \quad O_\ell = \sum_{i=1}^{n} I\left(\frac{t_i}{m_i} \in \mathcal{A}_\ell\right).$$

represent the expected and observed counts in the ℓth interval, respectively, for $\ell = 1, \ldots, r$. Sutradhar et al. (2008) showed that $X(\boldsymbol{\theta}) \sim \chi^2_{r-1}$ when all parameters are known and

$X(\hat{\boldsymbol{\theta}}) \sim \chi^2_{r-1-q}$ when $\boldsymbol{\theta} \in \Theta \subseteq \mathbb{R}^q$ is estimated by maximizing the likelihood under the given intervals (the *grouped* likelihood). In practice, it is more natural to work with the usual (*ungrouped*) likelihood

$$L_u(\boldsymbol{\theta}) = \prod_{i=1}^{n} f_i(y_i \mid m_i, \boldsymbol{\theta}).$$

There is a "recovery" of degrees of freedom in the GOF statistic when the ungrouped MLE is used, so that $X(\hat{\boldsymbol{\theta}})$ follows a χ^2_ν distribution with ν between $r-1-q$ and $r-1$. Although (Sutradhar et al., 2008) specifically discusses the random-clumped binomial distribution, the theory is given for general binomial models with varying m_i. A number of regularity conditions are assumed for the χ^2 distributions to hold.

In liu of checking these regularity conditions, the parametric bootstrap may be used to verify the distribution of the test statistic and determine a p-value for the GOF test. Suppose $\hat{\boldsymbol{\theta}}$ is the MLE computed from the (ungrouped) likelihood. The parametric bootstrap consists of drawing B samples

$$Y_i^{(b)} \overset{\text{ind}}{\sim} f_i(y_i \mid m_i, \hat{\boldsymbol{\theta}}), \quad \text{for } i = 1, \ldots, n \text{ and } b = 1, \ldots, B.$$

The test statistic $X^{(b)}$ is computed using the observations $(y_1^{(b)}, \ldots, y_n^{(b)})$ and the MLE $\hat{\boldsymbol{\theta}}^{(b)}$ from the bth bootstrap sample. The theoretical distribution of $X(\hat{\boldsymbol{\theta}})$ can be studied through the empirical distribution of $X^{(1)}, \ldots, X^{(B)}$, and a p-value can be computed using $\frac{1}{B}\sum_{b=1}^{B} I\left(X^{(b)} \geq X(\hat{\boldsymbol{\theta}})\right)$. Notice that $X(\hat{\boldsymbol{\theta}})$ remains fixed throughout the bootstrap procedure.

The selection of intervals $\mathcal{A}_\ell$ is left up to the analyst, but it is suggested to follow the rule of thumb that all $E_\ell(\boldsymbol{\theta}) \geq 5$ to ensure that the distribution theory holds. Common choices include equal width intervals or intervals having equal probability.

5.2 R Code

First we will write code to count the number of observations in each interval. Suppose we select the intervals $\mathcal{A}_1 = [0, 1/10], \mathcal{A}_2 = (1/10, 2/10], \ldots, \mathcal{A}_{10} = (9/10, 10/10]$.

```
> source("/home/araim/R/rcb.R")
> n <- 50
> m <- 20
> y <- r.rcb(n, Pi = 1/4, rho = 0.4, m = m)
> ( gof.breaks <- seq(0,1, 0.1) )
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
> ( Intervals <- cut(y/m, gof.breaks, include.lowest = TRUE) )
 [1] (0.4,0.5] (0.1,0.2] (0.5,0.6] [0,0.1]   (0.1,0.2] (0.8,0.9] (0.5,0.6] (0.2,0.3]
 [9] [0,0.1]   (0.4,0.5] (0.1,0.2] (0.2,0.3] (0.1,0.2] (0.2,0.3] (0.1,0.2] (0.1,0.2]
[17] (0.5,0.6] [0,0.1]   (0.3,0.4] (0.2,0.3] (0.6,0.7] (0.6,0.7] (0.2,0.3] (0.1,0.2]
[25] (0.1,0.2] (0.1,0.2] (0.6,0.7] (0.6,0.7] (0.3,0.4] (0.1,0.2] (0.1,0.2] (0.4,0.5]
[33] (0.4,0.5] (0.5,0.6] (0.2,0.3] [0,0.1]   (0.1,0.2] (0.2,0.3] (0.1,0.2] (0.1,0.2]
[41] (0.6,0.7] (0.6,0.7] (0.2,0.3] [0,0.1]   [0,0.1]   (0.1,0.2] (0.8,0.9] (0.5,0.6]
[49] (0.1,0.2] (0.1,0.2]
10 Levels: [0,0.1] (0.1,0.2] (0.2,0.3] (0.3,0.4] (0.4,0.5] (0.5,0.6] ... (0.9,1]
> table(Intervals)
Intervals
  [0,0.1] (0.1,0.2] (0.2,0.3] (0.3,0.4] (0.4,0.5] (0.5,0.6] (0.6,0.7] (0.7,0.8]
        6        17         8         2         4         5         6         0
(0.8,0.9]   (0.9,1]
        2         0
```

The `cut` function determines the interval for each sample proportion `y/m`. The breaks $(0, 1/10, \ldots, 10/10)$ along with the option `include.lowest = TRUE` ensure that intervals are set up as desired. The `table` function then counts the frequency of each interval among the entries of `y/m`. Putting parentheses around a statement causes the result to be printed; e.g. `(x <- 10)` is a shorthand for `print(x <- 10)`.

Computing the expected counts is a bit more involved, so we will proceed in smaller steps. First recall that the count for the ℓth interval is

$$E_\ell(\boldsymbol{\theta}) = \sum_{i=1}^{n} \sum_{t=0}^{m_i} \mathrm{P}(t \mid m_i, \boldsymbol{\theta}) I\left(\frac{t}{m_i} \in \mathcal{A}_\ell\right)$$

for $\ell = 1, \ldots, r$. For a given m_i, and assuming RCB$(m_i, \pi = 1/4, \rho = 0.2)$ as the null distribution, here is code to compute the inner summation.

```
1   m <- 20
2   gof.breaks <- seq(0, 1, 0.1)
3   K <- length(gof.breaks) - 1
4
5   # Note: Workaround for breaks 2, ..., K to obtain the correct expected counts
6   gof.breaks[-c(1,K+1)] <- gof.breaks[-c(1,K+1)] + .Machine$double.eps
7
8   tt <- seq(0, m)
9   ff <- d.rcb(tt, Pi = 1/4, rho = 0.2, m = m)
10  Int <- findInterval(x = tt / m, vec = gof.breaks, rightmost.closed = TRUE)
11  res <- aggregate(ff ~ Int, FUN = sum)
```

Line 9 computes the density at each sample point t_i while line 10 finds the index of the interval to which t_i/m_i belongs. Line 11 then sums the densities which belong to the same interval, so that have a table of interval indices and their total mass. Note the adjustment on line 6, where a tiny positive number (machine epsilon) is added to all breaks, except the two outer ones. This adjustment seems to be necessary with this code; without it, density is sometimes credited to the wrong intervals. The results for `ff`, `Int`, and `res` are shown below.

```
> ff
 [1] 8.656052e-03 4.335643e-02 1.034539e-01 1.571105e-01 1.723972e-01 1.495814e-01 1.129277e-01
 [8] 8.238295e-02 6.154710e-02 4.547484e-02 3.080895e-02 1.809498e-02 8.939284e-03 3.650751e-03
[15] 1.214836e-03 3.237482e-04 6.743129e-05 1.057650e-05 1.175127e-06 8.246400e-08 2.748787e-09
> Int
 [1]  1  1  1  2  2  3  3  4  4  5  5  6  6  7  7  8  8  9  9 10 10
> res
   Int           ff
1    1 1.554664e-01
2    2 3.295078e-01
3    3 2.625091e-01
4    4 1.439301e-01
5    5 7.628379e-02
6    6 2.703427e-02
7    7 4.865588e-03
8    8 3.911795e-04
9    9 1.175163e-05
10  10 8.521279e-08
```

Now we can compute the expected counts for a sample of size n

```
n <- 50
m <- rpois(n, lambda = 20)
gof.breaks <- seq(0, 1, 0.1)
f <- function(x, i) { d.rcb(x, Pi = 1/4, rho = 0.2, m = m[i]) }

K <- length(gof.breaks) - 1
gof.breaks[-c(1,K+1)] <- gof.breaks[-c(1,K+1)] + .Machine$double.eps
Ex <- numeric(K)

for (i in 1:n)
{
    tt <- seq(0, m[i])
    ff <- f(tt, i)
    Int <- findInterval(x = tt / m[i], vec = gof.breaks, rightmost.closed = TRUE)
    res <- aggregate(ff ~ Int, FUN = sum)
    Ex[res$Int] <- Ex[res$Int] + res$ff
}
```

Notice that we now let the number of trials `m[i]` vary with `i`. We have also replaced `d.rcb` with a general function `f` in the loop; this will allow us to reuse the code for other densities besides RCB. When computing the expected counts, we only need `f(x,i)`, the density for the `i`th subject evaluated at `x`. Running this code once obtains the following result.

```
> Ex
 [1] 5.609029e+00 1.537637e+01 1.410603e+01 7.786037e+00 5.041360e+00 1.585618e+00 4.098304e-01
 [8] 7.783251e-02 7.276444e-03 6.245881e-04
```

The full GOF procedure is given as Listing 2.

File: /home/araim/R/gof-binomial.R

```
gof.binomial <- function(y, m, f, gof.breaks, qq)
{
    if (length(m) == 1) m <- rep(m, length(y))
    K <- length(gof.breaks) - 1
    Ob <- gof.binomial.obs(y, m, gof.breaks)
    Ex <- numeric(K)
    n <- length(y)

    gof.breaks[-c(1,K+1)] <- gof.breaks[-c(1,K+1)] + .Machine$double.eps

    for (i in 1:n)
    {
        tt <- seq(0, m[i])
        ff <- f(tt, i)
        Int <- findInterval(x = tt / m[i], vec = gof.breaks, rightmost.closed = TRUE)
        res <- aggregate(ff ~ Int, FUN = sum)
        Ex[res$Int] <- Ex[res$Int] + res$ff
    }

    tab <- cbind(Ob, Ex)
    X <- sum( (Ob - Ex)^2 / Ex )
    df.low <- K - 1 - qq
    df.high <- K - 1
    pvalue.low <- 1 - pchisq(X, df.low)
    pvalue.high <- 1 - pchisq(X, df.high)

    brak <- c("[", rep("(", K-1))
    labels <- sprintf("%s%0.04f,%0.04f]", brak, gof.breaks[-(K+1)], gof.breaks[-1])
    rownames(tab) <- labels

    res <- list(tab = tab, X = X, df.low = df.low, df.high = df.high,
        pvalue.low = pvalue.low, pvalue.high = pvalue.high, f = f,
        breaks = gof.breaks, m = m)
    class(res) <- "gof.binomial"
    return(res)
}
```

Listing 2: Function for the binomial GOF procedure.

After the observed and expected counts are computed, we compose a table with their values, and compute the test statistic. We can then compute the degrees of freedom for the χ^2 test statistics when all parameters are known (`df.high`), and based on the grouped likelihood when all parameters are estimated (`df.low`). The number of estimated parameters `qq` is an argument to be provided by the user. The associated p-values are also computed. Now let us define a `print.gof.binomial` function so that the result will be displayed nicely.

```
print.gof.binomial <- function(gof.out)
{
    printf("GOF for model:\n")
    print(gof.out$f)
    printf("--\n")
    print(round(gof.out$tab, 4))
    printf("--\n")
    printf("X = %0.4f\n", gof.out$X)
    printf("DF in [%d, %d]\n", gof.out$df.low, gof.out$df.high)
    printf("p-value in [%f, %f]\n", gof.out$pvalue.low, gof.out$pvalue.high)
}
```

Suppose the `gof.binomial` and `print.gof.binomial` functions are stored in a file `gof.R` so that we can call it as follows.

```
source("/home/araim/R/gof.R")

n <- 100
m <- rpois(n, lambda = 20)
y <- r.rcb(n, Pi = 1/4, rho = 0.2, m)

extra.tx <- function(theta) { list(log.Pi = log(theta$Pi), rho.sq = theta$rho^2) }
fit.out <- fit.rcb.mle(y, m, extra.tx)

p.hat <- fit.out$estimates["Pi",1]
rho.hat <- fit.out$estimates["rho",1]

gof.breaks <- c(0, 0.1, 0.2, 0.3, 0.4, 1)
f <- function(x, i) { d.rcb(x, Pi = Pi.hat, rho = rho.hat, m = m[i]) }
gof.out <- gof.binomial(y, m, f, gof.breaks, qq = 2)
```

```
> gof.out
GOF for model:
function(x, i) { d.rcb(x, Pi = Pi.hat, rho = rho.hat, m = m[i]) }
--
                Ob      Ex
[0.0000,0.1000]  6  7.7699
(0.1000,0.1500] 11 10.3685
(0.1500,0.2000] 20 17.2156
(0.2000,0.2500] 24 18.7116
(0.2500,0.3000]  9 16.0526
(0.3000,0.4000] 21 21.3084
(0.4000,1.0000]  9  8.5735
--
X = 5.5108
DF in [4, 6]
p-value in [0.238785, 0.480157]
```

Here the GOF statistic indicates a very good fit, which is expected because the data were drawn from RCB. Next we will give code to compute the p-value and sampling distribution of the GOF test statistic based on the parametric bootstrap. Listing 3 gives the complete code as `gof-parboot.R`. A terminal session calling the script is given below, and the resulting plot is given as Figure 1.

```
> source("gof-parboot.R", print.eval = TRUE)
Starting bootstrap rep 1
Starting bootstrap rep 2
...
Starting bootstrap rep 500
> p.value.boot
[1] 0.53
```

File: /home/araim/R/gof-parboot.R

```r
source("gof.R")

# --------- Generate the data ---------
n <- 100
m <- rpois(n, lambda = 20)
y <- r.rcb(n, Pi = 1/4, rho = 0.2, m)

# --------- Fit the MLE for the observed data ---------
extra.tx <- function(theta) { }
fit.out <- fit.rcb.mle(y, m, extra.tx)
Pi.hat <- fit.out$estimates["Pi",1]
rho.hat <- fit.out$estimates["rho",1]

# --------- Compute GOF for the observed data ---------
gof.breaks <- c(0, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 1)
f <- function(x,i) { d.rcb(x, Pi = Pi.hat, rho = rho.hat, m = m[i]) }
gof.out <- gof.binomial(y, m, f, gof.breaks, qq = 2)

# --------- Parametric bootstrap for GOF statistic ---------
B <- 500
X.boot <- numeric(B)

for (b in 1:B)
{
    printf("Starting bootstrap rep %d\n", b)
    y.boot <- r.rcb(n, Pi.hat, rho.hat, m)

    fit.out.boot <- fit.rcb.mle(y.boot, m, extra.tx)
    Pi.hat.boot <- fit.out.boot$estimates["Pi",1]
    rho.hat.boot <- fit.out.boot$estimates["rho",1]

    f.boot <- function(x,i)
    {
        d.rcb(x, Pi = Pi.hat.boot, rho = rho.hat.boot, m = m[i])
    }
    gof.out.boot <- gof.binomial(y.boot, m, f.boot, gof.breaks, qq = 2)
    X.boot[b] <- gof.out.boot$X
}

p.value.boot <- mean(X.boot > gof.out$X)

# --------- Plot the bootstrap dist'n of the GOF statistic ---------
plot(ecdf(X.boot))
curve(pchisq(x, df = gof.out$df.low), add = TRUE, col = "red", lwd = 2)
curve(pchisq(x, df = gof.out$df.high), add = TRUE, col = "green", lwd = 2)
mynames <- c("ECDF of X.boot",
    sprintf("chisq df = %d", gof.out$df.low),
    sprintf("chisq df = %d", gof.out$df.high))
mycol <- c("black", "red", "green")
legend("bottomright", mynames, col = mycol, lwd = 2)
```

Listing 3: Code for parametric bootstrap of simulated data using RCB model.

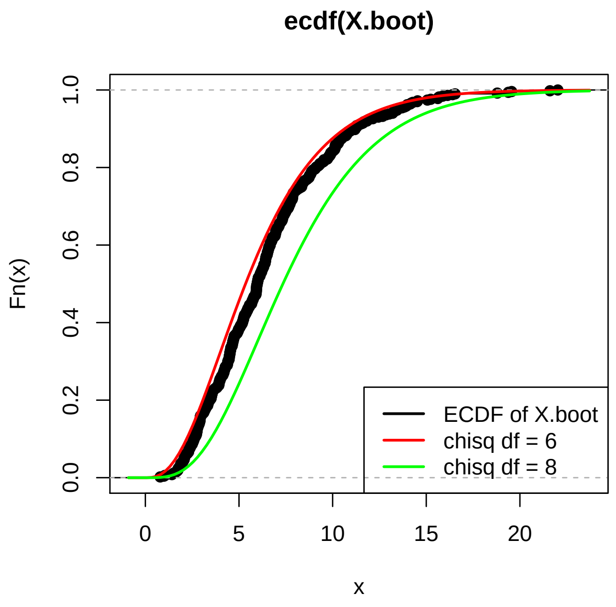


Figure 1: Empirical CDF of bootstrapped GOF test statistics for simulated data.

6 Example Data Analyses

In this section, we repeat some of the example data analyses in OMSAS. Using built-in capabilities of R, along with our numerical MLE framework, we focus on examples involving likelihood-based analysis of binomial/multinomial data with overdispersion. We first present an example involving simple linear regression, then progress to a binomial regression. We then move to examples involving RCB, BB, binomial finite mixtures, and Mixture Link. Here we demonstrate the goodness-of-fit test statistic from Section 5 for binomial data with varying number of trials. We then proceed to multinomial analysis using DM and RCM. Our final example involves BB and RCB models with a random effect.

The reader may repeat the programming steps in this section by first installing the `OverdispersionModelsInR` package, loading it using the following command, and obtaining the datasets as discussed in Section 1.2.

```
> library(OverdispersionModelsInR)
```

6.1 Analysis of Gain in Weight Data with Linear Regression

The Gain in Weight data are analyzed in chapter 2, pp. 19–22 of OMSAS using both ordinary least squares (OLS) and the MLE under the simple linear regression model. This is a good first example for R as well, before taking on more exotic likelihoods. First let us read the data.

```
> gain.in.weight <- read.table("/home/araim/data/gain-in-weight.dat", header = TRUE)
> tail(gain.in.weight)
   Initial Gain
10      48  118
11      57  107
12      59  106
13      46   82
14      45  103
15      65  104
```

The `tail` function is useful for showing the last few rows of a table or matrix. The function `lm` can be used to obtain the OLS estimate.

```
> lm.out <- lm(Gain ~ Initial, data = gain.in.weight)
> summary(lm.out)

Call:
lm(formula = Gain ~ Initial, data = gain.in.weight)

Residuals:
    Min      1Q  Median      3Q     Max
-32.373 -10.168  -1.308  14.717  35.948

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  54.9502    31.9768   1.718   0.1094
Initial       1.0641     0.5259   2.024   0.0641 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 19.29 on 13 degrees of freedom
Multiple R-squared:  0.2395,    Adjusted R-squared:  0.181
F-statistic: 4.095 on 1 and 13 DF,  p-value: 0.06408
```

The syntax Gain ∼ Initial says that Gain is the response in the linear model, and Initial is the covariate. The summary function returns a standard table of estimates. Many other functions are available to manipulate the lm.out object. Here are some examples.

```
> coef(lm.out)                ## Coefficients
(Intercept)     Initial
  54.950183    1.064092
> vcov(lm.out)                ## Estimated covariance of coefficients
            (Intercept)     Initial
(Intercept)  1022.51319 -16.6101936
Initial       -16.61019   0.2765293
> confint(lm.out, level=0.95)  ## Confidence intervals for coefficients
                   2.5 %     97.5 %
(Intercept) -14.13140706 124.031774
Initial      -0.07196023   2.200145
> AIC(lm.out)
[1] 135.2016
> BIC(lm.out)
[1] 137.3257
> anova(lm.out)
Analysis of Variance Table

Response: Gain
          Df Sum Sq Mean Sq F value  Pr(>F)
Initial    1 1522.9 1522.86  4.0947 0.06408 .
Residuals 13 4834.9  371.91
---
Signif. codes:  0 ’***’ 0.001 ’**’ 0.01 ’*’ 0.05 ’.’ 0.1 ’ ’ 1
> resid(lm.out)               ## Residuals
          1           2           3           4           5           6           7
 19.8451968  35.9479032  22.1787945  -4.0520968  -0.6930207  -6.7957272 -32.3725587
          8           9          10          11          12          13          14
 -1.3084663  17.4606424  11.9733816  -8.6034500 -11.7316348 -21.8984336   0.1656588
         15
-20.1161892
```

To obtain the MLE, we can use our numerical MLE framework.

```
y <- gain.in.weight$Gain
X <- model.matrix(~ Initial, data = gain.in.weight)
extra.tx <- function(theta) { list(sigmasq = theta$sigma^2) }
var.names <- c("Intercept", "Initial", "sigma", "sigmasq")
fit.out <- fit.normal.x.mle(y, X, extra.tx, var.names)
```

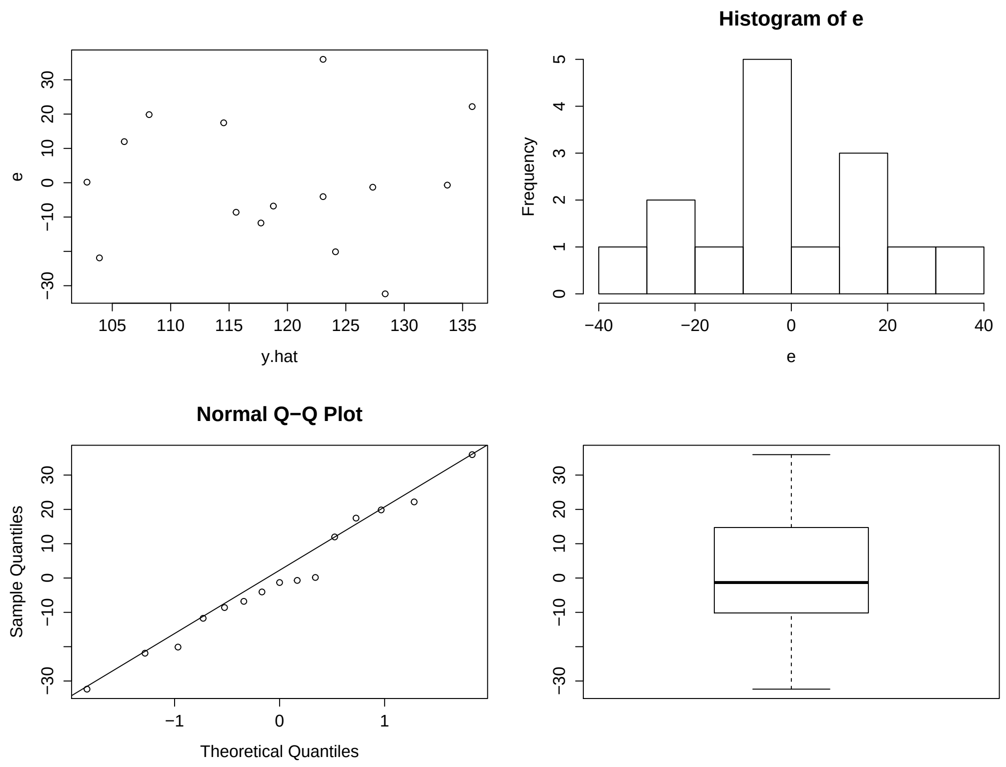
```
> tail(X)
   (Intercept) gain.in.weight$Initial
10           1                     48
11           1                     57
12           1                     59
13           1                     46
14           1                     45
15           1                     65
> fit.out
Fit for model:
y[i] ~indep~ N(mu[i], sigma^2)
mu[i] = x[i]^T Beta
--- Parameter Estimates ---
          Estimate      SE  t-val P(|t|>t-val)  Gradient
Intercept  54.9506 29.7687 1.8459       0.0847 1.647E-06
Initial     1.0641  0.4895 2.1736       0.0462    0.0001
sigma      17.9534  3.2778 5.4772    6.372E-05    0.0001
--- Additional Estimates ---
        Estimate       SE  t-val P(|t|>t-val) Gradient
sigmasq 322.3249 117.6963 2.7386       0.0152   0.0047
--
```

```
Degrees of freedom = 15
LogLik = -64.6008
AIC = 135.2016
AICC = 137.3834
BIC = 137.3257
```

Here we can see that the covariate matrix `X`, constructed with the help of `model.matrix`, contains an intercept in the first column and the variable `Initial` in the second column. One difference to note between our output and the `SAS PROC GLIMMIX` output given by OMSAS is that we have used n degrees of freedom to compute per-variable statistics, while `SAS GLIMMIX` uses with $n-2$ degrees of freedom. The two produce very similar results unless the sample size is very small. As in OMSAS, let us prepare some plots of the residuals based on the MLE.

```
Beta.hat <- fit.out$theta.hat$Beta
y.hat <- X %*% Beta.hat
e <- y - y.hat

par(mfrow = c(2,2))    # Create a 2x2 plot container
plot(y.hat, e)         # Predicted vs. residuals
hist(e)                # Histogram for residuals
qqnorm(e)              # Normal Q-Q plot with reference line added
qqline(e)
boxplot(e)             # Boxplot for residuals
```

The plots are specified with a minimal number of decorations. Some differences can be seen between our plots and those in Plot 2.1 of OMSAS. The latter feature different intervals for the histogram, different shading options, and more descriptive labels. Of course, with more work, the R plots can be decorated with these features as well.



6.2 Analysis of Pyrethrins Data with Binomial Regression

The Pyrethrins data are analyzed in chapter 2, pp. 28–39 of OMSAS using binomial regression models: first the logistic regression model (with logistic link), then the probit model (with probit link). Binomial regression is an example of a Generalized Linear Model (GLM), a generalization of linear regression which can model some kinds of non-Normal observations such as categories and counts. Both the logistic and probit links can be fitted in `R` using the `glm` function. First let us read the data.

```
> pyrethrins <- read.table("/home/araim/data/pyrethrins.dat", header = TRUE)
> tail(pyrethrins)
   dose   m   t
6   140 469 400
7   160 550 495
8   180 542 499
9   200 479 450
10  250 497 476
11  300 453 442
```

In this example, `t` is considered a binomial response out of `m` trials, and the probability is modeled as

$$T_i \overset{\text{ind}}{\sim} \text{Bin}(m_i, p_i), \quad p_i = G(\beta_0 + \beta_1 \log(\text{dose})),$$

where G is an appropriate inverse link function. There several equivalent ways to fit this model using `glm`.

```
## These give the same result
logit.out <- glm( cbind(t,m-t) ~ log(dose), data = pyrethrins, family = binomial )
logit.alt.out <- glm(t/m ~ log(dose), data = pyrethrins, weights = m, family = binomial)
```

```
> summary(logit.out)

Call:
glm(formula = cbind(t, m - t) ~ log(dose), family = binomial,
    data = pyrethrins)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-1.65828  -0.51149   0.08672   0.46433   1.28430

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -10.3279     0.3429  -30.12   <2e-16 ***
log(dose)     2.4582     0.0747   32.91   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1529.2572  on 10  degrees of freedom
Residual deviance:    6.3585  on  9  degrees of freedom
AIC: 73.914

Number of Fisher Scoring iterations: 3
```

Notice that the `log(dose)` can be specified right in the model statement. The `cbind(t, m-t)` statement creates an $n \times 2$ matrix with number of successes in the first column and number of failures in the second column. The argument `family = binomial` carries out a binomial regression rather than a linear (Normal) regression, which is the default. The alternative syntax incorporates the number of trials `m`, perhaps less intuitively, by treating the proportions `t/m` as responses and weighting them using `weights = m`. Note that `glm` assumes Normality for the standardized estimates, while `PROC GLIMMIX` assumes the t distribution with $n-2$ degrees of freedom. We can specify the probit link instead as follows.

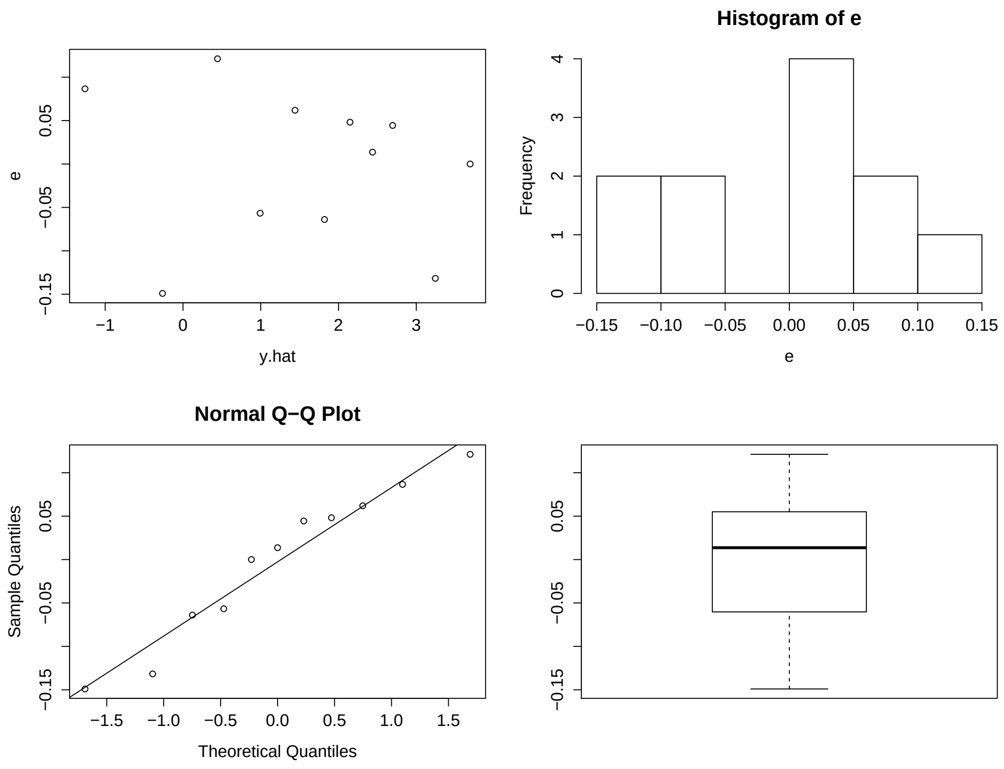
```
> probit.out <- glm( cbind(t,m-t) ~ log(dose), data = pyrethrins, family = binomial(link = "probit") )
> summary(probit.out)

Call:
glm(formula = cbind(t, m - t) ~ log(dose), family = binomial(link = "probit"),
    data = pyrethrins)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-2.00507  -0.59772   0.00041   0.34960   1.57292

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -5.91393    0.18816  -31.43   <2e-16 ***
log(dose)    1.40900    0.04033   34.93   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1529.257  on 10  degrees of freedom
Residual deviance:   11.644  on  9  degrees of freedom
AIC: 79.2

Number of Fisher Scoring iterations: 4
```

OMSAS computes the quantity LD50, which is dose level at which the probability of death is 1/2. For both the logit and probit models, LD50 is computed as $-\beta_0/\beta_1$. We can calculate an estimate in R as follows.

```
> Beta.hat <- coef(logit.out)
> printf("Estimate for LD50 under logit is %f", -Beta.hat[1] / Beta.hat[2])
Estimate for LD50 under logit is 4.201334
> Beta.hat <- coef(probit.out)
> printf("Estimate for LD50 under probit is %f", -Beta.hat[1] / Beta.hat[2])
Estimate for LD50 under probit is 4.197263
```

We can also produce similar residual plots as OMSAS using the logit model, keeping the amount of decoration to a minimum. Several types of residuals are available from the `resid` function. We select `type = working` to match the SAS analysis.

```
y.hat <- predict(logit.out)
e <- resid(logit.out, type = "working")

par(mfrow = c(2,2))    # Create a 2x2 plot container
plot(y.hat, e)         # Predicted vs. residuals
hist(e)                # Histogram for residuals
qqnorm(e)              # Normal Q-Q plot with reference line added
qqline(e)
boxplot(e)             # Boxplot for residuals
```



6.3 Analysis of Ossification Data with Generalized Linear Overdispersion Models

The Ossification data in chapter 4, pp. 103–112 of OMSAS are used as an example of binomial data with extra variation. The book considers several models using binomial, RCB, beta-binomial likelihoods. While the binomial model is considered to be a GLM, RCB and BB are said to be examples of Generalized Linear Overdispersion Models (GLOMs). RCB and BB do not fit into the exponential family framework, therefore general procedures developed for GLMs cannot be used with them. However, regressions can be linked to their parameters without any conceptual difficulty. We can use the `glm` function in base `R` to fit binomial regressions, and our numerical MLE framework can handle the others. First we will read the data.

```
> ossification <- read.table("/home/araim/data/ossification.dat", header = TRUE)
> tail(ossification)
   litter    group oss size
76     76 PHT+TCPO   0    4
77     77 PHT+TCPO   0    6
78     78 PHT+TCPO   0    7
79     79 PHT+TCPO   6    6
80     80 PHT+TCPO   1    6
81     81 PHT+TCPO   1    7
> levels(ossification$group)
[1] "Control"  "PHT"      "PHT+TCPO" "TCPO"
```

The variable `oss` is considered to be a number of successes out of `size` trials. The variable `group` is treated as a categorical covariate (whose posible values are shown by the `levels` function) which influences the probability of success. Let $\text{PHT}_i = 1$ if the ith subject received the PHT treatment, and 0 otherwise. Similarly, let TCPO_i be the indicator for TCPO. The models under consideration will be:

- Logistic: $T_i \overset{\text{ind}}{\sim} \text{Bin}(m_i, \pi_i)$
- RCB: $T_i \overset{\text{ind}}{\sim} \text{RCB}(m_i, \pi_i, \rho)$
- BB: $T_i \overset{\text{ind}}{\sim} \text{BB}(m_i, \pi_i, \rho)$
- RCB-Reg: $T_i \overset{\text{ind}}{\sim} \text{RCB}(m_i, \pi_i, \rho_i)$
- BB-Reg: $T_i \overset{\text{ind}}{\sim} \text{BB}(m_i, \pi_i, \rho_i)$

All models have a common regression on π_i given by

$$g(\pi_i) = \beta_0 + \beta_1 \text{TCPO}_i + \beta_2 \text{PHT}_i + \beta_3 (\text{TCPO}_i \cdot \text{PHT}_i)$$

The "-reg" models have an additional regression on the overdispersion parameter ρ_i given by

$$g(\rho_i) = \alpha_0 + \alpha_1 \text{TCPO}_i + \alpha_2 \text{PHT}_i + \alpha_3 (\text{TCPO}_i \cdot \text{PHT}_i)$$

First we transform the data to prepare for analysis. `X` will represent the matrix of covariates for π and `Z` will represent the covariates for ρ.

```
tcpo <- ossification$group %in% c("TCPO", "PHT+TCPO")
pht <- ossification$group %in% c("PHT", "PHT+TCPO")
both <- ossification$group %in% c("PHT+TCPO")

X <- cbind(1, tcpo, pht, both)
colnames(X) <- c("Intercept", "TCPO", "PHT", "PHT+TCPO")

Z <- X

y <- ossification$oss
m <- ossification$size
```

The line `ossification$group %in% c("TCPO", "PHT+TCPO")` returns a vector containing `TRUE` and `FALSE`, if the corresponding entry of `ossification$group` had value `"TCPO"` or `"PHT+TCPO"`. We first use the `glm` function to fit standard logistic regression. Notice that `X` can be used in the model formula to specify the covariates.

```
> glm.out <- glm(cbind(y,m-y) ~ X -1, family = binomial)
> summary(glm.out)

Call:
glm(formula = cbind(y, m - y) ~ X - 1, family = binomial)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.6797  -1.5294   0.1009   1.5095   4.0580

Coefficients:
           Estimate Std. Error z value Pr(>|z|)
XIntercept   0.8323     0.1365   6.097 1.08e-09 ***
XTCPO       -0.8481     0.2239  -3.788 0.000152 ***
XPHT        -2.1094     0.2505  -8.422  < 2e-16 ***
XPHT+TCPO    1.0453     0.4107   2.545 0.010921 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 393.81  on 81  degrees of freedom
```

```
Residual deviance: 290.43  on 77  degrees of freedom
AIC: 409.78

Number of Fisher Scoring iterations: 4
```

Now let us fit the BB and RCB models without the additional regression on ρ. We specify a number of additional parameters to match the SAS output of OMSAS. The estimates of π are given for each of the four treatment groups:

$$\pi_{\text{Control}} = G(\beta_0), \quad \pi_{\text{TCPO}} = G(\beta_0 + \beta_1), \quad \pi_{\text{PHT}} = G(\beta_0 + \beta_2),$$
$$\pi_{\text{PHT+TCPO}} = G(\beta_0 + \beta_1 + \beta_2 + \beta_3).$$

Also, the log-odds-ratio of PHT vs. Control is requested, both when TCPO is present and absent. In our models, these log-odds-ratios are parameterized by $\beta_2 + \beta_3$ and β_2 respectively.

```
var.names <- c(colnames(X), "rho", "Pi Control", "Pi PHT", "Pi TCPO",
    "Pi PHT+TCPO", "Log-odds-ratio PHT vs. Control, TCPO Present",
    "Log-odds-ratio PHT vs. Control, TCPO Absent", "rho.sq")
extra.tx <- function(theta)
{
    list(Pi.control = plogis(theta$Beta[1]),
        Pi.TCPO = plogis(sum(theta$Beta[1:2])),
        Pi.PHT = plogis(sum(theta$Beta[c(1,3)])),
        Pi.PHT_TCPO = plogis(sum(theta$Beta[1:4])),
        log.odds.tcpo = theta$Beta[3],
        log.odds.notcpo = sum(theta$Beta[3:4]),
        rho.sq = theta$rho^2)
}

fit.rcb.x.out <- fit.rcb.x.mle(y, m, X = X, extra.tx = extra.tx, var.names = var.names)
fit.bb.x.out <- fit.bb.x.mle(y, m, X = X, extra.tx = extra.tx, var.names = var.names)
```

The results for the RCB model are given below, followed by the BB model, both of which match closely to the OMSAS analysis.

```
> fit.rcb.x.out
Fit for model:
y[i] ~indep~ RCB(m[i], Pi[i], rho)
logit(Pi[i]) = x[i]^T Beta
--- Parameter Estimates ---
          Estimate     SE   t-val P(|t|>t-val)   Gradient
Intercept   0.6392 0.2266  2.8204       0.0060     0.0003
TCPO       -0.9456 0.3711 -2.5481       0.0127  5.367E-05
PHT        -1.5291 0.3956 -3.8657       0.0002  4.795E-05
PHT+TCPO    0.6161 0.6678  0.9226       0.3589     0.0001
rho         0.5831 0.0417 13.9926    0.000E+00 -4.272E-05
--- Additional Estimates ---
                                 Estimate     SE   t-val P(|t|>t-val)   Gradient
Pi Control                         0.6546 0.0512 12.7741    0.000E+00  5.989E-05
Pi PHT                             0.4240 0.0737  5.7510    1.517E-07  7.779E-05
Pi TCPO                            0.2911 0.0634  4.5946    1.573E-05  6.456E-05
Pi PHT+TCPO                        0.2280 0.0826  2.7623       0.0071  9.019E-05
Log-OR PHT vs. Control, w/TCPO    -1.5291 0.3956 -3.8657       0.0002  4.795E-05
Log-OR PHT vs. Control, w/o TCPO  -0.9129 0.5608 -1.6278       0.1074     0.0002
rho.sq                             0.3400 0.0486  6.9963    6.856E-10 -4.982E-05
--
Degrees of freedom = 81
LogLik = -152.5267
AIC = 315.0534
AICC = 315.8534
BIC = 327.0257
```

```
> fit.bb.x.out
Fit for model:
y[i] ~indep~ BB(m[i], Pi[i], rho)
logit(Pi[i]) = x[i]^T Beta
--- Parameter Estimates ---
           Estimate     SE   t-val P(|t|>t-val)   Gradient
Intercept    0.7043 0.2341  3.0087       0.0035    -0.0002
TCPO        -0.7822 0.4017 -1.9474       0.0550    -0.0001
PHT         -1.6917 0.4018 -4.2102    6.563E-05    -0.0001
PHT+TCPO     0.6769 0.6902  0.9808       0.3296  3.822E-05
rho          0.5808 0.0466 12.4609    0.000E+00 -3.082E-05
--- Additional Estimates ---
                                 Estimate     SE   t-val P(|t|>t-val)   Gradient
Pi Control                         0.6691 0.0518 12.9117    0.000E+00 -3.410E-05
Pi PHT                             0.4805 0.0816  5.8870    8.548E-08 -7.051E-05
Pi TCPO                            0.2714 0.0628  4.3211    4.376E-05 -5.811E-05
Pi PHT+TCPO                        0.2511 0.0883  2.8434       0.0056 -7.222E-05
Log-OR PHT vs. Control, w/TCPO    -1.6917 0.4018 -4.2102    6.563E-05    -0.0001
Log-OR PHT vs. Control, w/o TCPO  -1.0148 0.5727 -1.7720       0.0802    -0.0001
rho.sq                             0.3374 0.0541  6.2304    1.969E-08 -3.580E-05
--
Degrees of freedom = 81
LogLik = -153.2876
AIC = 316.5751
AICC = 317.3751
BIC = 328.5474
```

Now we proceed to fit the RCB-reg and BB-reg models. Following OMSAS, we report the π and ρ^2 values specific to each group.

```
var.names <- c(sprintf("X:%s", colnames(X)), sprintf("Z:%s", colnames(Z)),
    "Pi Control", "Pi PHT", "Pi TCPO", "Pi PHT+TCPO",
    "rho.sq Control", "rho.sq PHT", "rho.sq TCPO", "rho.sq PHT+TCPO",
    "Log-odds-ratio PHT vs. Control, TCPO Present",
    "Log-odds-ratio PHT vs. Control, TCPO Absent")
extra.tx <- function(theta)
{
    list(Pi.control = plogis(theta$Beta[1]),
         Pi.TCPO = plogis(sum(theta$Beta[1:2])),
         Pi.PHT = plogis(sum(theta$Beta[c(1,3)])),
         Pi.PHT_TCPO = plogis(sum(theta$Beta[1:4])),
         rho.sq.control = plogis(theta$Gamma[1])^2,
         rho.sq.TCPO = plogis(sum(theta$Gamma[1:2]))^2,
         rho.sq.PHT = plogis(sum(theta$Gamma[c(1,3)]))^2,
         rho.sq.PHT_TCPO = plogis(sum(theta$Gamma[1:4]))^2,
         log.odds.tcpo = theta$Beta[3],
         log.odds.notcpo = sum(theta$Beta[3:4]))
}

fit.rcb.xz.out <- fit.rcb.xz.mle(y, m, X = X, Z = Z, extra.tx = extra.tx, var.names = var.names)
fit.bb.xz.out <- fit.bb.xz.mle(y, m, X = X, Z = Z, extra.tx = extra.tx, var.names = var.names)
```

```
> fit.rcb.xz.out
Fit for model:
y[i] ~indep~ RCB(m[i], Pi[i], rho[i])
logit(Pi[i]) = x[i]^T Beta
logit(rho[i]) = z[i]^T Gamma
--- Parameter Estimates ---
             Estimate     SE   t-val P(|t|>t-val)   Gradient
X:Intercept    0.5669 0.2455  2.3093       0.0235     0.0008
X:TCPO        -0.8712 0.3924 -2.2204       0.0292     0.0005
X:PHT         -1.8405 0.3413 -5.3931    6.691E-07     0.0007
X:PHT+TCPO     1.4055 0.7080  1.9852       0.0505 -1.846E-05
Z:Intercept    0.5160 0.2603  1.9821       0.0509    -0.0020
Z:TCPO        -0.1909 0.4006 -0.4765       0.6350 -8.893E-05
```

```
Z:PHT        -1.8772 0.9942 -1.8882       0.0626    -0.0003
Z:PHT+TCPO    3.3734 1.1949  2.8231       0.0060  1.668E-05
--- Additional Estimates ---
                                Estimate     SE   t-val P(|t|>t-val) Gradient
Pi Control                        0.6380 0.0567 11.2542    0.000E+00   0.0002
Pi PHT                            0.4245 0.0748  5.6766    2.070E-07   0.0003
Pi TCPO                           0.2186 0.0405  5.3982    6.551E-07   0.0002
Pi PHT+TCPO                       0.3231 0.1180  2.7384       0.0076   0.0004
rho.sq Control                    0.3921 0.0763  5.1384    1.878E-06  -0.0006
rho.sq PHT                        0.3371 0.0861  3.9145       0.0002  -0.0006
rho.sq TCPO                       0.0416 0.0636  0.6547       0.5145  -0.0001
rho.sq PHT+TCPO                   0.7408 0.1215  6.0961    3.508E-08  -0.0005
Log-OR PHT vs. Control, w/TCPO   -1.8405 0.3413 -5.3931    6.691E-07   0.0007
Log-OR PHT vs. Control, w/o TCPO -0.4350 0.6203 -0.7013       0.4851   0.0006
--
Degrees of freedom = 81
LogLik = -143.9084
AIC = 303.8168
AICC = 305.8168
BIC = 322.9724
```

```
> fit.bb.xz.out
Fit for model:
y[i] ~indep~ BB(m[i], Pi[i], phi[i])
logit(Pi[i]) = x[i]^T Beta
logit(rho[i]) = z[i]^T Gamma
--- Parameter Estimates ---
            Estimate     SE   t-val P(|t|>t-val)   Gradient
X:Intercept   0.6822 0.2424  2.8136       0.0061  1.004E-05
X:TCPO       -0.7616 0.4108 -1.8537       0.0674    -0.0001
X:PHT        -1.9546 0.3433 -5.6942    1.924E-07 -6.801E-05
X:PHT+TCPO    1.2565 0.7173  1.7518       0.0836  8.751E-05
Z:Intercept   0.4200 0.2771  1.5156       0.1335    -0.0002
Z:TCPO       -0.0257 0.4766 -0.0538       0.9572     0.0004
Z:PHT        -1.6527 0.8882 -1.8606       0.0664  5.165E-05
Z:PHT+TCPO    2.5935 1.1336  2.2878       0.0248  6.586E-05
--- Additional Estimates ---
                                Estimate     SE   t-val P(|t|>t-val)   Gradient
Pi Control                        0.6642 0.0541 12.2837    0.000E+00  2.238E-06
Pi PHT                            0.4802 0.0828  5.8000    1.234E-07 -2.409E-05
Pi TCPO                           0.2188 0.0415  5.2678    1.114E-06 -9.911E-06
Pi PHT+TCPO                       0.3149 0.1155  2.7260       0.0079 -1.661E-05
rho.sq Control                    0.3642 0.0800  4.5506    1.859E-05 -5.685E-05
rho.sq PHT                        0.3568 0.1114  3.2018       0.0020  5.966E-05
rho.sq TCPO                       0.0509 0.0666  0.7652       0.4464 -1.145E-05
rho.sq PHT+TCPO                   0.6268 0.1535  4.0823       0.0001  8.491E-05
Log-OR PHT vs. Control, w/TCPO   -1.9546 0.3433 -5.6942    1.924E-07 -6.801E-05
Log-OR PHT vs. Control, w/o TCPO -0.6981 0.6298 -1.1084       0.2710  1.950E-05
--
Degrees of freedom = 81
LogLik = -147.8976
AIC = 311.7951
AICC = 313.7951
BIC = 330.9507
```

6.4 Analysis of Hiroshima Data with Generalized Linear Overdispersion Models

The Hiroshima data presented in chapter 5, pp. 130–138, of OMSAS are used as an example of binomial data with extra variation, as well as to demonstrate the goodness-of-fit test for binomial data with varying m_i. The authors consider the RCB and BB models. We will also consider logistic regression, the finite mixture of binomial regressions, and the Mixture Link binomial model which was developed after OMSAS was published. First we read the data.

```
> hiroshima <- read.table("/home/araim/data/hiroshima.dat", header = TRUE)
> tail(hiroshima)
      m t t65d_gamma t65d_neutron
643 100 0          0            0
644 100 2          0            0
645 100 1          0            0
646  80 2         79           41
647  65 0          0            0
648  40 1          0            0
```

```
z <- scale(hiroshima$t65d_gamma + hiroshima$t65d_neutron)
X <- model.matrix(~ z + I(z^2))
Z <- model.matrix(~ z + I(z^2))
y <- hiroshima$t
m <- hiroshima$m
n <- length(y)
d <- ncol(X)
```

The total radiation dose (gamma + neutron) is considered as the covariate in this example, and the `scale` function is used to standardize it so that it has mean 0 and standard deviation 1 across all subjects. Denote $g = G^{-1}$ as the logit link function. We will consider the following models:

$$\begin{aligned}
&\text{Logistic: } Y_i \overset{\text{ind}}{\sim} \text{Bin}(m_i, \pi_i),\\
&\qquad g(\pi_i) = \beta_0 + \beta_1 z_i + \beta_2 z_i^2,\\
&\text{RCB-Reg: } Y_i \overset{\text{ind}}{\sim} \text{RCB}(m_i, \pi_i, \rho_i),\\
&\qquad g(\pi_i) = \beta_0 + \beta_1 z_i + \beta_2 z_i^2,\\
&\qquad g(\rho_i) = \gamma_0 + \gamma_1 z_i + \gamma_2 z_i^2\\
&\text{BB-Reg: } Y_i \overset{\text{ind}}{\sim} \text{BB}(m_i, \pi_i, \rho_i),\\
&\qquad g(\pi_i) = \beta_0 + \beta_1 z_i + \beta_2 z_i^2,\\
&\qquad g(\rho_i) = \gamma_0 + \gamma_1 z_i + \gamma_2 z_i^2\\
&\text{MixtureJ2: } Y_i \overset{\text{ind}}{\sim} \text{BinMix}_2(m_i, p_{i1}, p_{i2}, \pi),\\
&\qquad g(p_{i1}) = \beta_{10} + \beta_{11} z_i + \beta_{12} z_i^2,\\
&\qquad g(p_{i2}) = \beta_{20} + \beta_{21} z_i + \beta_{22} z_i^2\\
&\text{MixLinkJ2: } Y_i \overset{\text{ind}}{\sim} \text{MixLink}_2(m_i, p_i, \boldsymbol{\pi}, \kappa),\\
&\qquad g(p_i) = \beta_0 + \beta_1 z_i + \beta_2 z_i^2.
\end{aligned}$$

Note that all models have a regression on the probability of success, and the two "-Reg" models also have a regression on the overdispersion parameter. The `glm` function can be used to fit the Logsitic model.

```
> glm.out <- glm(cbind(y,m) ~ X -1, family = binomial)
> summary(glm.out)

Call:
glm(formula = cbind(y, m) ~ X - 1, family = binomial)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-5.1038  -1.6474  -0.3271   0.5029   6.4331
```

```
Coefficients:
             Estimate Std. Error z value Pr(>|z|)
X(Intercept) -3.09153    0.02452 -126.11   <2e-16 ***
Xz            1.23634    0.03409   36.27   <2e-16 ***
XI(z^2)      -0.29878    0.01577  -18.95   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 66549.2  on 648  degrees of freedom
Residual deviance:  1723.5  on 645  degrees of freedom
AIC: 3327.4

Number of Fisher Scoring iterations: 5
> qq <- 3; n <- nrow(X)
> printf("AICC = %f", -2*logLik(glm.out) + 2*3*n / (n-3-1))
AICC = 3327.470059
> printf("BIC = %f", BIC(glm.out))
BIC = 3340.854464
```

Notice the model formula ∼`X -1`, which says to use `X` as the design matrix for the regression, and not to include an additional intercept term. Next we will fit the RCB-Reg model using the numerical MLE framework.

```
> fit.rcb.xz.out <- fit.rcb.xz.mle(y, m, X = X, Z = Z)
> fit.rcb.xz.out
Fit for model:
y[i] ~indep~ RCB(m[i], Pi[i], rho[i])
logit(Pi[i]) = x[i]^T Beta
logit(rho[i]) = z[i]^T Gamma
--- Parameter Estimates ---
       Estimate     SE    t-val P(|t|>t-val)   Gradient
Beta1   -3.0699 0.0338 -90.8803    0.000E+00     0.0006
Beta2    1.3010 0.0444  29.2966    0.000E+00 -9.447E-05
Beta3   -0.3705 0.0244 -15.1551    0.000E+00    -0.0282
Gamma1  -2.3526 0.0965 -24.3778    0.000E+00     0.0391
Gamma2   0.9333 0.1569   5.9492    4.409E-09     0.0760
Gamma3  -0.2366 0.0565  -4.1906    3.168E-05     0.2122
--
Degrees of freedom = 648
LogLik = -1546.6120
AIC = 3105.2239
AICC = 3105.3550
BIC = 3132.0672
```

Here is the fit for the BB-Reg model.

```
> fit.bb.xz.out <- fit.bb.xz.mle(y, m, X = X, Z = Z)
> fit.bb.xz.out
Fit for model:
y[i] ~indep~ BB(m[i], Pi[i], phi[i])
logit(Pi[i]) = x[i]^T Beta
logit(rho[i]) = z[i]^T Gamma
--- Parameter Estimates ---
       Estimate     SE    t-val P(|t|>t-val) Gradient
Beta1   -3.0146 0.0445 -67.7443    0.000E+00   0.0205
Beta2    1.3594 0.0564  24.0898    0.000E+00  -0.0101
Beta3   -0.3449 0.0332 -10.3743    0.000E+00  -0.0222
Gamma1  -1.8610 0.0737 -25.2374    0.000E+00  -0.0053
Gamma2   0.7992 0.1109   7.2098    1.567E-12   0.0072
Gamma3  -0.1610 0.0525  -3.0659       0.0023   0.0060
--
Degrees of freedom = 648
LogLik = -1429.6054
```

```
AIC = 2871.2107
AICC = 2871.3418
BIC = 2898.0541
```

For the BinMixJ2 model, in addition to the likelihood parameters, let us estimate $\boldsymbol{\beta}_{\text{avg}} = \pi_1(\beta_{10}, \beta_{11}, \beta_{12}) + \pi_2(\beta_{20}, \beta_{21}, \beta_{22})$, the averaged coefficients weighted by the proportions in each subpopulation. The averaged coefficients should intuitively be similar to the estimates from MixLinkJ2, which is a more formal way of estimating the regression for the mixed population.

```
# ------------------- Binomial Finite Mixture, J = 2 -------------------
X.g <- list(X, X)
extra.tx <- function(theta)
{
    Beta.mat <- matrix(unlist(theta$Beta.g), 2, ncol(X), byrow = TRUE)
    list(Beta.avg = t(Beta.mat) %*% theta$Pi)
}
fit.binmix.x.out <- fit.binmix.x.mle(y, m, X.g, extra.tx)
print(fit.binmix.x.out)

# ------------------- MixlinkJ2 -------------------
Beta.init <- coef(glm.out)
Pi.init <- fit.binmix.x.out$theta.hat$Pi
kappa.init <- 1.5
phi.init <- c(Beta.init, qlogis(Pi.init), log(kappa.init))
fit.mixlink.x.out <- fit.mixture.link.x.mle(y, m, X, J = 2, phi.init = phi.init)
```

```
> fit.binmix.x.out
Fit for model:
y[i] ~indep~ BinMix_2(m[i], p[i], Pi)
logit(p[i,j]) = x[i]^T Beta[j]
--- Parameter Estimates ---
          Estimate     SE    t-val P(|t|>t-val) Gradient
Beta.g.11  -2.4141 0.0619 -39.0195    0.000E+00   0.0055
Beta.g.12   1.3869 0.0599  23.1416    0.000E+00   0.0130
Beta.g.13  -0.3536 0.0261 -13.5424    0.000E+00   0.0018
Beta.g.21  -3.5394 0.0773 -45.8059    0.000E+00   0.0050
Beta.g.22   1.4240 0.0847  16.8147    0.000E+00   0.0307
Beta.g.23  -0.4414 0.0413 -10.6770    0.000E+00   0.0448
Pi1         0.3781 0.0469   8.0636    3.553E-15  -0.0011
Pi2         0.6219 0.0469  13.2607    0.000E+00   0.0011
--- Additional Estimates ---
          Estimate     SE    t-val P(|t|>t-val) Gradient
Beta.avg1  -3.1139 0.0408 -76.3716    0.000E+00   0.0039
Beta.avg2   1.4100 0.0591  23.8437    0.000E+00   0.0241
Beta.avg3  -0.4082 0.0281 -14.5285    0.000E+00   0.0284
--
Degrees of freedom = 648
LogLik = -1500.8893
AIC = 3017.7786
AICC = 3018.0039
BIC = 3053.5697
```

The MixLinkJ2 model requires a lot of computation to fit with the numerical MLE framework. The estimates for $\boldsymbol{\pi}$ and $\boldsymbol{\beta}_{\text{avg}}$ from BinMixJ2 should intuitively provide good starting values for MixLinkJ2, to reduce the work needed by the optimization routine.

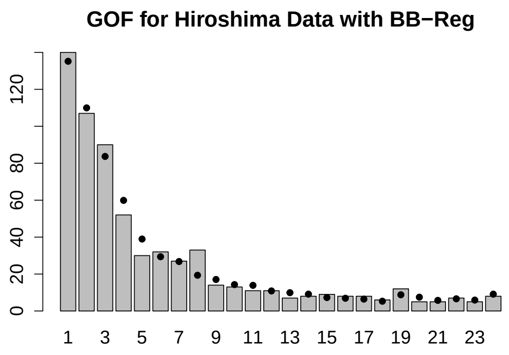
```
Beta.init <- fit.binmix.x.out$xi.hat$Beta.avg
Pi.init <- fit.binmix.x.out$theta.hat$Pi
kappa.init <- 1
phi.init <- c(Beta.init, qlogis(Pi.init), log(kappa.init))
fit.mixlink.x.out <- fit.mixture.link.x.mle(y, m, X, J = 2, phi.init = phi.init)
```

```
> fit.mixlink.x.out
Fit for model:
y[i] ~indep~ MixLink_2(m[i], p[i], Pi, kappa)
logit( E(Y[i]) ) = x[i]^T Beta
Using method: imhof
Elapsed time: 126.250378 sec
--- Parameter Estimates ---
      Estimate     SE    t-val P(|t|>t-val) Gradient
Beta1  -3.0062 0.0441 -68.1105    0.000E+00   0.0032
Beta2   1.3655 0.0563  24.2719    0.000E+00   0.0171
Beta3  -0.3382 0.0314 -10.7670    0.000E+00  -0.0148
Pi1     0.3297 0.0176  18.7739    0.000E+00  -0.0003
Pi2     0.6703 0.0176  38.1740    0.000E+00   0.0003
kappa   1.6293 0.2487   6.5513    1.163E-10   0.0026
--
Degrees of freedom = 648
LogLik = -1433.3312
AIC = 2878.6624
AICC = 2878.7934
BIC = 2905.5057
```

We would also like to apply the GOF test to these models. Listing 4 shows the R code and results for the GOF test under the BB-Reg, including a plot of the observed vs. expected counts. The results for the other four models are not shown, but the code is similar. Table 1 gives a summary of the GOF results from the five models, along with AIC, AICC, and BIC. The BB-Reg and MixLinkJ2 models give the best overall fit.

```
# ------------------ BB-Reg ------------------
Beta.hat <- fit.bb.xz.out$theta.hat$Beta
Gamma.hat <- fit.bb.xz.out$theta.hat$Gamma
Pi.hat <- plogis(X %*% Beta.hat)
rho.hat <- plogis(Z %*% Gamma.hat)
f <- function(x,i) { d.beta.binom(x, Pi.hat[i], rho.hat[i], m[i]) }

gof.breaks <- c(seq(0, 0.1783, 0.0099), 0.1980, 0.2178, 0.2376, 0.2673, 0.3069, 1)
gof.bb.xz.out <- gof.binomial(y, m, f, gof.breaks, qq = ncol(X) + ncol(Z))

K <- nrow(gof.bb.xz.out$tab)
coords <- barplot(gof.bb.xz.out$tab[,"Ob"], names.arg = 1:K)
points(coords, gof.bb.xz.out$tab[,"Ex"], pch = 19)
title("GOF for Hiroshima Data with BB-Reg")
```

```
> gof.bb.xz.out
GOF for model:
function(x,i) { d.beta.binom(x, Pi.hat[i],
rho.hat[i], m[i]) }
--
                     Ob       Ex
[0.0000,0.0099] 140 135.1971
(0.0099,0.0198] 107 109.9622
(0.0198,0.0297]  90  83.6866
(0.0297,0.0396]  52  59.8884
(0.0396,0.0495]  30  38.9632
(0.0495,0.0594]  32  29.3928
(0.0594,0.0693]  27  26.7869
(0.0693,0.0792]  33  19.3542
(0.0792,0.0891]  14  17.0632
(0.0891,0.0990]  13  14.3010
(0.0990,0.1089]  11  13.8918
(0.1089,0.1188]  11  10.8686
(0.1188,0.1287]   7   9.9161
(0.1287,0.1386]   8   9.1336
(0.1386,0.1485]   9   7.2917
(0.1485,0.1584]   8   6.9002
(0.1584,0.1683]   8   6.4359
(0.1683,0.1782]   6   5.3048
(0.1782,0.1980]  12   8.7917
(0.1980,0.2178]   5   7.4826
(0.2178,0.2376]   5   5.7533
(0.2376,0.2673]   7   6.5965
(0.2673,0.3069]   5   5.9168
(0.3069,1.0000]   8   9.1205
--
X = 19.3959
DF in [17, 23]
p-value in [0.306285, 0.678020]
```



Listing 4: GOF code and output for Hiroshima data under BB-Reg model.

The p-value for the GOF test is given as a range, based on two chi-square distributions which form an upper and lower bound for the large sample distribution of $X(\hat{\boldsymbol{\theta}})$. We can find a specific p-value using the parametric bootstrap. Listing 5 gives the R code to carry out the bootstrap procedure. Figure 2 plots the bootstrapped GOF test statistics. For the bootstrapped p-value we obtain `p.value.boot = 0.555`. The parametric bootstrap can be repeated for the other models as well, but note that it is significantly more time consuming for the MixLinkJ2 model because of the work involved in computing the likelihood.

Table 1: Model comparison statistics.

Model	LogLik	q	AIC	AICC	BIC	GOF statistic	GOF df range	GOF p-value
Logistic	-1814.189	3	3634.400	3327.470	3647.799	110.38	[17,20]	$< 10^{-13}$
RCB-Reg	-1546.612	6	3105.224	3105.355	3132.067	63.96	[18,22]	$< 10^{-5}$
BB-Reg	-1429.605	6	2871.211	2871.342	2898.054	19.40	[17,23]	> 0.3063
BinMixJ2	-1500.889	8	3017.779	3018.004	3053.570	37.34	[15,23]	< 0.0012
MixLinkJ2	-1433.331	5	2876.662	2878.793	2905.506	19.50	[18,23]	> 0.3615

File: /home/araim/R/gof-parboot-hiroshima-bbreg.R

```r
set.seed(1234)

# --------- Parametric bootstrap for GOF statistic ---------
# Uses MLE (Pi.hat, rho.hat) computed from the observed data
B <- 200
n <- length(y); d1 <- nrow(X); d2 <- nrow(Z)
gof.out <- gof.bb.xz.out
X.boot <- numeric(B)

for (b in 1:B)
{
    printf("Starting bootstrap rep %d\n", b)
    y.boot <- r.beta.binom(n, Pi.hat, rho.hat, m)
    fit.boot <- fit.bb.xz.mle(y.boot, m, X, Z, phi.init = c(Beta.hat, Gamma.hat))

    Beta.hat.boot <- fit.boot$theta.hat$Beta
    Gamma.hat.boot <- fit.boot$theta.hat$Gamma
    Pi.hat.boot <- plogis(X %*% Beta.hat.boot)
    rho.hat.boot <- plogis(Z %*% Gamma.hat.boot)

    f.boot <- function(x, i) { d.beta.binom(x, Pi.hat.boot[i], rho.hat.boot[i], m[i]) }
    gof.out.boot <- gof.binomial(y.boot, m, f.boot, gof.breaks, qq = ncol(X) + ncol(Z))
    X.boot[b] <- gof.out.boot$X
}

p.value.boot <- mean(X.boot > gof.out$X)
printf("Finished bootstrap. p.value.boot = %g\n", p.value.boot)

# --------- Plot the bootstrap dist'n of the GOF statistic ---------
pdf("gof-bbreg-boot-ecdf.pdf", width = 5, height = 5)
plot(ecdf(X.boot))
curve(pchisq(x, df = gof.out$df.low), add = TRUE, col = "red", lwd = 2)
curve(pchisq(x, df = gof.out$df.high), add = TRUE, col = "green", lwd = 2)
mynames <- c("ECDF of X.boot",
    sprintf("chisq df = %d", gof.out$df.low),
    sprintf("chisq df = %d", gof.out$df.high))
mycol <- c("black", "red", "green")
legend("bottomright", mynames, col = mycol, lwd = 2)
dev.off()
```

Listing 5: Parametric bootstrap for the GOF test statistic under BB-Reg model.

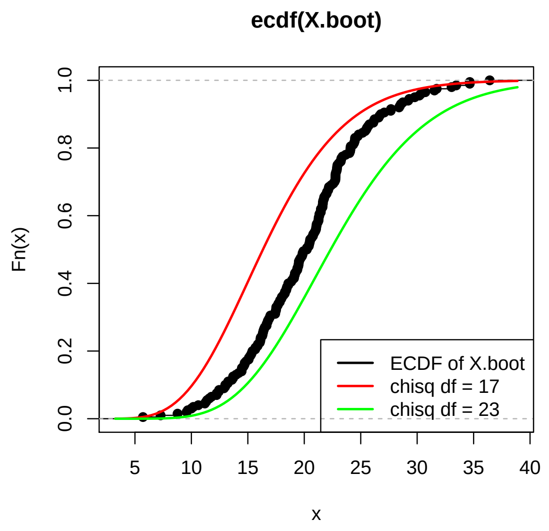


Figure 2: Empirical CDF of bootstrapped GOF test statistics under the BB-Reg model.

6.5 Analysis of Forest Pollen Data with Multinomial Overdispersion Models

The Forest Pollen data are considered in chapter 7 of OMSAS on pp. 228–231 to demonstrate two multinomial models with extra variation: RCM and DM. For this analysis we will use the scoring framework developed in Section 4. First we will read the data.

```
> pollen <- read.table("/home/araim/data/forest-pollen.dat", head = TRUE)
> tail(pollen)
   t1 t2 t3 t4
68 80  0 14  6
69 95  1  3  1
70 84  0 14  2
71 81  2  9  8
72 85  3  9  3
73 94  3  3  0
```

We can see that there are $n = 73$ multinomial counts with $k = 4$ categories. These correspond to counts of pollen for four arbor types: pine, fir, oak and alder. Each observation has a common number of trials $m = 100$. We will fit the RCM and DM models via the scoring framework, but first let us exact the data into the necessary format.

```
> x <- t(as.matrix(pollen))
> m <- colSums(x)
> n <- ncol(x)
> Data <- list(x = x, m = m, n = n)
```

The `k x n` matrix `x` has `n` columns which represent the observations. In this example, for both RCM and DM, we make use of the hybrid algorithm consisting of approximate scoring iterations then Newton-Raphson iterations. This keeps the number of iterations from getting too large, and gives more accurate standard errors than using approximate scoring alone. The fitting of RCM is shown in Listing 6, while DM is shown in Listing 7. Notice here that π_4, the probability of observing alder, is not explicitly computed because it is not included in $\boldsymbol{\theta}$, but could be computed as an extra quantity of interest if desired.

```
> rcm <- rcm.scoring(k = nrow(x))
> theta.init <- c(1/4,1/4,1/4, 0.5)
> var.names <- c("Pi1", "Pi2", "Pi3", "rho")
> out.nr.hyb <- fit.family.nr.hybrid(theta.init, rcm, Data, var.names = var.names)
NR Hybrid: Doing Approx Scoring until warmup.tol = 0.0001
Approx Scoring: After iter 1: loglik = -730.042614, delta = Inf, estimates:
[1] 0.50657760 0.08533744 0.27684138 0.81698630
...
Approx Scoring: After iter 74: loglik = -534.217134, delta = 7.550291e-06, estimates:
[1] 0.86835346 0.01514326 0.08626722 0.08974689
NR Hybrid: Doing Newton-Raphson until tol = 1e-06
No hessian function was specified. Using a numerical one.
Newton-Raphson: After iter 1: loglik = -534.217134, delta = Inf, estimates:
[1] 0.86835528 0.01514305 0.08626789 0.08974337
Newton-Raphson: After iter 2: loglik = -534.217134, delta = 4.547474e-13, estimates:
[1] 0.86835528 0.01514305 0.08626789 0.08974338
```

```
> out.nr.hyb
Fit for model:
X[i,] ~ind~ RCM_k(m[i], Pi, rho)
--- Parameter Estimates ---
    Estimate     SE    t-val P(|t|>t-val)   Gradient
Pi1   0.8684 0.0048 180.4764    0.000E+00 -1.210E-11
Pi2   0.0151 0.0016   9.7539    7.327E-15 -7.001E-10
Pi3   0.0863 0.0041  21.1720    0.000E+00 -3.617E-10
rho   0.0897 0.0111   8.0909    9.502E-12 -3.116E-10
--
Degrees of freedom = 73
LogLik = -534.2171
AIC = 1076.4343
AICC = 1077.0225
BIC = 1085.5961
Iterations = 76
Tolerance = 4.54747e-13
```

Listing 6: Code and results for forest pollen data under RCM model.

```
> dm <- dm.scoring(k = nrow(x))
> theta.init <- c(1/4,1/4,1/4, 0.5)
> var.names <- c("Pi1", "Pi2", "Pi3", "rho")
> out.nr.hyb <- fit.family.nr.hybrid(theta.init, dm, Data, var.names = var.names)
NR Hybrid: Doing Approx Scoring until warmup.tol = 0.0001
Approx Scoring: After iter 1: loglik = -756.394254, delta = Inf, estimates:
[1] 0.58816376 0.02368668 0.26957957 0.69301450
...
Approx Scoring: After iter 23: loglik = -507.822166, delta = 9.676465e-05, estimates:
[1] 0.86211455 0.01642600 0.08879773 0.12798303
NR Hybrid: Doing Newton-Raphson until tol = 1e-06
No hessian function was specified. Using a numerical one.
Newton-Raphson: After iter 1: loglik = -507.822063, delta = Inf, estimates:
[1] 0.86211481 0.01642556 0.08879901 0.12783209
Newton-Raphson: After iter 2: loglik = -507.822063, delta = 2.242473e-10, estimates:
[1] 0.86211477 0.01642557 0.08879903 0.12783232
```

```
> out.nr.hyb
Fit for model:
X[i,] ~ind~ DM_k(m[i], Pi, rho)
--- Parameter Estimates ---
    Estimate     SE    t-val P(|t|>t-val)   Gradient
Pi1   0.8621 0.0065 131.9555    0.000E+00  1.065E-07
Pi2   0.0164 0.0022   7.4052    1.834E-10  2.320E-07
Pi3   0.0888 0.0053  16.6813    0.000E+00  3.053E-08
rho   0.1278 0.0112  11.4478    0.000E+00 -1.679E-06
--
Degrees of freedom = 73
LogLik = -507.8221
AIC = 1023.6441
AICC = 1024.2324
BIC = 1032.8060
Iterations = 25
Tolerance = 2.24247e-10
```

Listing 7: Code and results for forest pollen data under DM model.

6.6 Analysis of Diaper Data with Generalized Linear Overdispersion Mixed Models

The Diaper data are considered in chapter 10 of OMSAS on pp. 356–365. They are used in demonstrating binomial models with extra variation with random effects, namely RCB and BB. Such models are GLOMs with random effects, and are referred to as Generalized Linear Overdispersion Mixed Models (GLOMMs). Mixed effect RCB and BB models can be fit with our numerical MLE framework, but we must do some additional work in programming the likelihood so that the random effects are integrated out.

Note that the diaper dataset is simulated. OMSAS generates the data under RCB with a random effect, then fits RCB to this version of the generated data. The process is then repeated for BB. In this section, we will fit both models to the data generated under RCB. Therefore the results obtained by fitting RCB will match OMSAS, but the ones obtained under BB will be different. We now read the data.

```
> diaper <- read.table("/home/araim/data/diaper.dat", sep = ",", head = TRUE)
> tail(diaper)
    SubjId Sequence Period Product X t  m
795    398       BA      1       2 0 3 20
796    398       BA      2       1 1 0 20
797    399       BA      1       2 0 1 20
798    399       BA      2       1 1 0 20
799    400       BA      1       2 0 7 20
800    400       BA      2       1 1 5 20
```

In this example, a 2×2 crossover design is used to evaluate the quality of two diaper products, comparing the probability of a leak. Subjects (babies) are assigned to one of two sequences:

- AB uses product A the 1st period and product B the 2nd period, and
- BA uses product B the 1st period and product A the 2nd period.

Let m_{sij} denote the number of diaper usages for the jth subject for the ith period in the sth sequence. Sequence $s = 1$ corresponds to the AB group and $s = 2$ is the BA group. For the AB group, $i = 1$ corresponds to product A and $i = 2$ corresponds to product B. On the other hand, for the BA group, product B is used when $i = 1$ and product A is used when $i = 2$. There are $n = 400$ subjects in the experiment, so that $j = 1, \ldots, 400$. Of the m_{sij} diaper usages, y_{sij} leakages are observed. In this simulated example $m_{sij} = 20$ for all observations, so we will use the notation m for simplicity. The data stucture is shown in Table 2.

Sequence	Period 1	Period 2
AB	$y_{111}, \ldots, y_{11n}$	$y_{121}, \ldots, y_{12n}$
BA	$y_{221}, \ldots, y_{22n}$	$y_{211}, \ldots, y_{21n}$

Table 2: 2×2 crossover design for diaper experiment.

We will consider fitting three binomial-type models with a subject-specific random effect $u_j \overset{\text{iid}}{\sim} \text{N}(0, \sigma_b^2)$,

$$\begin{aligned}
Y_{sij} &\overset{\text{ind}}{\sim} \text{Bin}(m, \pi_{sij}), \quad g(\pi_{sij}) = \beta_0 + \beta_1 x_{sij} + u_j, \\
Y_{sij} &\overset{\text{ind}}{\sim} \text{RCB}(m, \pi_{sij}, \rho), \quad g(\pi_{sij}) = \beta_0 + \beta_1 x_{sij} + u_j, \\
Y_{sij} &\overset{\text{ind}}{\sim} \text{BB}(m, \pi_{sij}, \rho), \quad g(\pi_{sij}) = \beta_0 + \beta_1 x_{sij} + u_j.
\end{aligned}$$

Here, our covariate x_{sij} is defined as the indicator for whether product B was currently in use; that is,

$$x_{sij} = \begin{cases} 1 & \text{if } s = 1 \text{ and } i = 2, \\ 1 & \text{if } s = 2 \text{ and } i = 1, \\ 0 & \text{otherwise.} \end{cases}$$

Fitting the binomial model is possible through the `glmer` function in the `lme4` package.

```
> library(lme4)
> glmm.out <- glmer(t/m ~ X + (1 | SubjId), data = diaper, weights = m, family = binomial)
> summary(glmm.out)
Generalized linear mixed model fit by maximum likelihood (Laplace Approximation) ['glmerMod']
 Family: binomial  ( logit )
Formula: t/m ~ X + (1 | SubjId)
   Data: diaper
Weights: m

     AIC      BIC   logLik deviance df.resid
  3630.1   3644.2  -1812.1   3624.1      797

Scaled residuals:
    Min      1Q  Median      3Q     Max
-3.2083 -0.7464 -0.5020  0.6306  3.4479

Random effects:
 Groups Name        Variance Std.Dev.
 SubjId (Intercept) 2.631    1.622
Number of obs: 800, groups: SubjId, 400

Fixed effects:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -2.80444    0.09883 -28.377  < 2e-16 ***
X            0.33956    0.05313   6.391 1.65e-10 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Correlation of Fixed Effects:
  (Intr)
X -0.306
> confint(glmm.out)
Computing profile confidence intervals ...
                2.5 %     97.5 %
.sig01       1.476215  1.7870490
(Intercept) -3.002895 -2.6148784
X            0.235568  0.4438659
```

Now we fit RCB and BB through our numerical MLE framework. The results are given in Listings 8 and 9. We obtain confidence intervals for all parameters after computing the estimates. Note that we have used 400 as the degrees of freedom, while OMSAS reports 399 degrees of freedom. This is justified by treating the (y_{s1j}, y_{s2j}) as bivariate observations from $n = 400$ individuals. The code to fit the RCB and BB models is given as Listings 10 and 11. We use the numerical MLE framework as in previous examples, where the likelihoods now do the work of integrating over the random effect. This is done numerically using quadrature. Notice that the likelihood is of the form

$$\begin{aligned} L(\boldsymbol{\theta}) &= \int_{-\infty}^{\infty} \prod_{s=1}^{2} \prod_{i=1}^{2} \prod_{j=1}^{n} f(y_{sij} \mid \pi_{sij}, \rho, m, u_j) \, \mathsf{N}(u_j \mid 0, \sigma_b^2) \, du_j \\ &= \prod_{s=1}^{2} \prod_{j=1}^{n} \int_{-\infty}^{\infty} f(y_{s1j} \mid \pi_{s1j}, \rho, m, u) f(y_{s2j} \mid \pi_{s2j}, \rho, m, u) \, \mathsf{N}(u \mid 0, \sigma_b^2) \, du, \end{aligned}$$

where f represents either the RCB or BB density and $\mathsf{N}(x \mid \mu, \sigma^2)$ is the Normal density. Every subject requires evaluation of a univariate integral, where the integrand involves both of his/her observations. We have used the `statmod` package to provide Q quadrature points $z_1, \ldots, z_Q$ from Normal$(0, \sigma_b^2)$ and corresponding weights $w_1, \ldots, w_Q$. The integral for the sth sequence of the jth subject can then approximated as

$$\int_{-\infty}^{\infty} h(u)\mathsf{N}(u \mid 0, \sigma_b^2)du \approx \sum_{\ell=1}^{Q} w_\ell h(z_\ell),$$

where $h(u) = f(y_{s1j} \mid \pi_{s1j}, \rho, m, u) f(y_{s2j} \mid \pi_{s2j}, \rho, m, u)$.

```
source("/home/araim/R/fit-diaper-rcb-mle.R")
source("/home/araim/R/fit-diaper-bb-mle.R")

extra.tx <- function(theta)
{
    list(sigma.b.sq = theta$sigma.b^2,
    Pi1 = plogis(theta$Beta[1] + theta$Beta[2]),
    Pi2 = plogis(theta$Beta[1]),
    OR = exp(theta$Beta[2]),
    rho.sq = theta$rho^2)
}

fit.rcb.mixeff.out <- fit.rcb.mixeff.mle(diaper, extra.tx, nGQ = 20)
fit.bb.mixeff.out <- fit.bb.mixeff.mle(diaper, extra.tx, nGQ = 20)
```

```
> fit.rcb.mixeff.out <- fit.rcb.mixeff.mle(diaper, extra.tx, phi.init = phi.init, nGQ = 20)
> print(fit.rcb.mixeff.out)
Fit for model:
y[s,i,j] ~ind~ RCB(m[s,i,j], Pi[s,i,j], rho),
logit(Pi[s,i,j]) = x[s,i,j]^T Beta + u[j]
u[j] ~iid~ N(0, sigma.b^2)
Likelihood computed by Gaussian Quadrature with nGQ = 20
Elapsed time: 5.20 sec
--- Parameter Estimates ---
        Estimate     SE    t-val P(|t|>t-val)  Gradient
Beta1    -2.5638 0.0939 -27.3163    0.000E+00 1.164E-05
Beta2     0.3771 0.0773   4.8802    1.533E-06   -0.0003
rho       0.3954 0.0150  26.3869    0.000E+00 7.262E-05
sigma.b   1.2672 0.0760  16.6806    0.000E+00    0.0006
--- Additional Estimates ---
           Estimate     SE   t-val P(|t|>t-val)   Gradient
sigma.b.sq   1.6059 0.1925  8.3403    1.332E-15     0.0015
Pi1          0.1010 0.0080 12.6159    0.000E+00 -2.754E-05
Pi2          0.0715 0.0062 11.4753    0.000E+00  7.729E-07
OR           1.4581 0.1127 12.9403    0.000E+00    -0.0005
rho.sq       0.1564 0.0119 13.1935    0.000E+00  5.744E-05
--
Degrees of freedom = 400
LogLik = -1577.7232
AIC = 3163.4464
AICC = 3163.5477
BIC = 3179.4123
```

```
> confint(fit.rcb.mixeff.out)
--- Parameter CIs (level 0.950000) ---
          Estimate         SE      Lower      Upper
Beta1   -2.5637645 0.09385489 -2.7482750 -2.3792540
Beta2    0.3771336 0.07727798  0.2252118  0.5290553
rho      0.3954422 0.01498629  0.3659804  0.4249039
sigma.b  1.2672234 0.07596976  1.1178735  1.4165733
--- Additional CIs (level 0.950000) ---
            Estimate          SE      Lower      Upper
sigma.b.sq 1.6058552 0.192541308 1.22733583 1.98437450
Pi1        0.1009575 0.008002372 0.08522552 0.11668944
Pi2        0.0715072 0.006231394 0.05925683 0.08375758
OR         1.4580991 0.112678952 1.23658212 1.67961600
rho.sq     0.1563745 0.011852419 0.13307368 0.17967532
--
Degrees of freedom = 400
t-quantile = 1.965912
```

Listing 8: Estimates and confidence intervals for RCB random effect model applied to diaper data.

```
> print(fit.bb.mixeff.out)
Fit for model:
y[s,i,j] ~ind~ BB(m[s,i,j], Pi[s,i,j], rho),
logit(Pi[s,i,j]) = x[s,i,j]^T Beta + u[j]
u[j] ~iid~ N(0, sigma.b^2)
Likelihood computed by Gaussian Quadrature with nGQ = 20
Elapsed time: 5.60 sec
--- Parameter Estimates ---
        Estimate     SE    t-val P(|t|>t-val) Gradient
Beta1    -2.4408 0.1000 -24.4115    0.000E+00  -0.0005
Beta2     0.3430 0.0858   3.9988    7.582E-05  -0.0004
rho       0.3474 0.0184  18.8632    0.000E+00   0.0003
sigma.b   1.1845 0.0824  14.3681    0.000E+00   0.0006
--- Additional Estimates ---
           Estimate     SE   t-val P(|t|>t-val)   Gradient
sigma.b.sq   1.4032 0.1953  7.1841    3.333E-12     0.0014
Pi1          0.1093 0.0092 11.9244    0.000E+00 -8.643E-05
Pi2          0.0801 0.0074 10.8726    0.000E+00 -3.448E-05
OR           1.4092 0.1209 11.6574    0.000E+00    -0.0006
rho.sq       0.1207 0.0128  9.4316    0.000E+00     0.0002
--
Degrees of freedom = 400
LogLik = -1635.1215
AIC = 3278.2430
AICC = 3278.3442
BIC = 3294.2088
```

```
> confint(fit.bb.mixeff.out)
--- Parameter CIs (level 0.950000) ---
          Estimate         SE      Lower      Upper
Beta1   -2.4407687 0.09998453 -2.6373295 -2.2442078
Beta2    0.3430266 0.08578207  0.1743865  0.5116666
rho      0.3473727 0.01841533  0.3111698  0.3835757
sigma.b  1.1845492 0.08244298  1.0224735  1.3466249
--- Additional CIs (level 0.950000) ---
            Estimate          SE      Lower      Upper
sigma.b.sq 1.40315679 0.195315537 1.01918356 1.78713001
Pi1        0.10931647 0.009167438 0.09129409 0.12733885
Pi2        0.08011625 0.007368623 0.06563018 0.09460231
OR         1.40920621 0.120884626 1.17155763 1.64685479
rho.sq     0.12066783 0.012793968 0.09551601 0.14581965
--
Degrees of freedom = 400
t-quantile = 1.965912
```

Listing 9: Estimates and confidence intervals for BB random effect model applied to diaper data.

File: /home/araim/R/fit-diaper-rcb-mle.R

```
library(statmod)

fit.rcb.mixeff.mle <- function(diaper, extra.tx, phi.init = NULL, nGQ = 20)
{
    X <- model.matrix(~X, data = diaper)
    d <- ncol(X)

    Data <- list(y = diaper$t, m = diaper$m, X = X, n = max(diaper$SubjId))

    if (is.null(phi.init))
        phi.init <- c(rep(0,d+2))

    theta.tx <- function(phi)
    {
        list(Beta = phi[1:d], rho = plogis(phi[1+d]), sigma.b = exp(phi[2+d]))
    }

    loglik <- function(phi, Data)
    {
        theta <- theta.tx(phi)
        quad.out <- gauss.quad.prob(n = nGQ, dist = "normal", mu = 0, sigma = theta$sigma.b)
        u <- quad.out$nodes
        w <- quad.out$weights

        ll <- numeric(Data$n)
        eta.fixed <- Data$X %*% theta$Beta

        # Each subject has two observations, which are consecutive in the dataset
        # These correspond to the two periods of the experiment
        for (j in 1:Data$n)
        {
            idx.p1 <- 2*j - 1
            idx.p2 <- 2*j
            mu.p1 <- plogis(eta.fixed[idx.p1] + u)
            mu.p2 <- plogis(eta.fixed[idx.p2] + u)
            GQ.points.p1 <- d.rcb(Data$y[idx.p1], Pi = mu.p1, theta$rho, Data$m[idx.p1])
            GQ.points.p2 <- d.rcb(Data$y[idx.p2], Pi = mu.p2, theta$rho, Data$m[idx.p2])
            GQ.points <- GQ.points.p1 * GQ.points.p2

            ll[j] <- log(sum(w * GQ.points))
        }

        return(sum(ll))
    }

    start <- Sys.time()
    fit.out <- fit.mle(phi.init, loglik, theta.tx, extra.tx = extra.tx, Data = Data)

    fit.out$description <- paste0(
        sprintf("y[s,i,j] ~ind~ RCB(m[s,i,j], Pi[s,i,j], rho),\n"),
        sprintf("logit(Pi[s,i,j]) = x[s,i,j]^T Beta + u[j]\n"),
        sprintf("u[s,j] ~iid~ N(0, sigma.b^2)\n"),
        sprintf("Likelihood computed by Gaussian Quadrature with nGQ = %d\n", nGQ),
        sprintf("Elapsed time: %0.02f sec", as.numeric(Sys.time() - start, units = "secs"))
    )
    return(fit.out)
}
```

Listing 10: Code to fit random effect RCB model to diaper data.

File: /home/araim/R/fit-diaper-bb-mle.R

```r
library(statmod)

fit.bb.mixeff.mle <- function(diaper, extra.tx, phi.init = NULL, nGQ = 20)
{
    X <- model.matrix(~X, data = diaper)
    d <- ncol(X)

    Data <- list(y = diaper$t, m = diaper$m, X = X, n = max(diaper$SubjId))

    if (is.null(phi.init))
        phi.init <- c(rep(0,d+2))

    theta.tx <- function(phi)
    {
        list(Beta = phi[1:d], rho = plogis(phi[1+d]), sigma.b = exp(phi[2+d]))
    }

    loglik <- function(phi, Data)
    {
        theta <- theta.tx(phi)
        quad.out <- gauss.quad.prob(n = nGQ, dist = "normal", mu = 0, sigma = theta$sigma.b)
        u <- quad.out$nodes
        w <- quad.out$weights

        ll <- numeric(Data$n)
        eta.fixed <- Data$X %*% theta$Beta

        # Each subject has two observations, which are consecutive in the dataset
        # These correspond to the two periods of the experiment
        for (j in 1:Data$n)
        {
            idx.p1 <- 2*j - 1
            idx.p2 <- 2*j
            mu.p1 <- plogis(eta.fixed[idx.p1] + u)
            mu.p2 <- plogis(eta.fixed[idx.p2] + u)
            GQ.points.p1 <- d.beta.binom(Data$y[idx.p1], Pi = mu.p1, theta$rho, Data$m[idx.p1])
            GQ.points.p2 <- d.beta.binom(Data$y[idx.p2], Pi = mu.p2, theta$rho, Data$m[idx.p2])
            GQ.points <- GQ.points.p1 * GQ.points.p2

            ll[j] <- log(sum(w * GQ.points))
        }

        return(sum(ll))
    }

    start <- Sys.time()
    fit.out <- fit.mle(phi.init, loglik, theta.tx, extra.tx = extra.tx, Data = Data)

    fit.out$description <- paste0(
        sprintf("y[s,i,j] ~ind~ BB(m[s,i,j], Pi[s,i,j], rho),\n"),
        sprintf("logit(Pi[s,i,j]) = x[s,i,j]^T Beta + u[j]\n"),
        sprintf("u[s,j] ~iid~ N(0, sigma.b^2)\n"),
        sprintf("Likelihood computed by Gaussian Quadrature with nGQ = %d\n", nGQ),
        sprintf("Elapsed time: %0.02f sec", as.numeric(Sys.time() - start, units = "secs"))
    )
    return(fit.out)
}
```

Listing 11: Code to fit random effect BB model to diaper data.

7 Notes on R Programming

7.1 Debugging

Debugging is an integral part of writing nontrivial programs. It is a rare occurrence to write code that works correctly in all circumstances on the first attempt. We often find that our new program produces errors or warnings, or gives an answer which appears incorrect. Sometimes the problem is difficult to pinpoint; errors may occur only in some random samples, and we may not be totally certain that an answer is incorrect. A few basic debugging techniques can help us track down the sources of these issues.

- Develop code "interactively". It is easy in R to write a few lines of code, run them, and examine the output. Doing this can help to prevent many obvious errors, and to quickly put together a working program.

- Use print statements to report important values. This is especially useful for longer programs that cannot be run interactively. For example, to see how the optimization is working on one of our `loglik` functions, we may want to print the result before returning it.

```
loglik <- function(phi, Data)
{
    theta <- theta.tx(phi)
    ll <- sum( d.rcb(Data$y, Pi = theta$Pi, rho = theta$rho, m = Data$m, log = TRUE) )
    printf("Pi=%f, rho=%f, ll=%f\n", theta$Pi, theta$rho, ll)
    return(ll)
}
```

```
> n <- 100; m <- 20
> Pi <- 1/2; rho <- 1/4
> y <- r.rcb(n, Pi, rho, m)
> fit.rcb.mle(y, m)
Pi=0.500000, rho=0.500000, ll=-314.103046
Pi=0.500250, rho=0.500000, ll=-314.104756
Pi=0.499750, rho=0.500000, ll=-314.101440
Pi=0.500000, rho=0.500250, ll=-314.228695
Pi=0.500000, rho=0.499750, ll=-313.977602
...
```

  In some cases, it may be appropriate to leave print statements in code for informational purposes, even after debugging is complete.

- R has an interactive debugger which can step through a program while it is running. There are several ways to start the debugger. For example, we can ask R to invoke the debugger any time a particular function is called. Note that this can be any function in R, not necessarily only ones that we have written. Suppose we want to use the debugger when the `optim` function is called. This can be accomplished by the command `debug(optim)`.

```
> debug(optim)
> fit.rcb.mle(y, m)
debugging in: optim(par = phi.init, fn = loglik, method = "L-BFGS-B", control = list(fnscale = -1,
    trace = 0), hessian = TRUE, Data = Data)
debug: {
    fn1 <- function(par) fn(par, ...)
... [optim function contents are shown] ...
}
```

Now the program is paused and we can look around using regular R commands.

```
Browse[2]> ls()
[1] "control" "fn"      "gr"      "hessian" "lower"   "method"  "par"     "upper"
Browse[2]> print(control)
$fnscale
[1] -1

$trace
[1] 0

Browse[2]>
```

Note that changes made to your workspace (e.g. changes to variables) within the debugging session may be discarded after exiting the debugger. We can stop the debugger by entering the command `Q`, or step to the next line of code by entering `n`. The debugger will continue to start each time `optim` is called in our current R session, until we enter `undebug(optim)`.

- Another way to invoke the debugger is to put a `browser` call in your program. R starts the debugger when it encounters this statement.

```
f <- function(x)
{
    z <- t(x) %*% x
    browser()
    return(z)
}
```

```
> x <- c(1,2,3)
> f(x)
Called from: f(x)
Browse[1]> ls()
[1] "x" "z"
Browse[1]> x
[1] 1 2 3
Browse[1]> z
     [,1]
[1,]   14
Browse[1]> Q
```

For more information about debugging in R, a useful web page is www.stats.uwo.ca/faculty/murdoch/software/debuggingR.

7.2 Writing Efficient R Code

It is very easy to write slow programs in R, but it is also possible to write efficient code by the following principle. Many R functions are implemented in C and are themselves very efficient. Take the `sum` function as a very simple example. You will see a huge performance improvement by calling `sum` on a large vector of numbers, rather than adding the numbers yourself in a loop.

```
> x <- rnorm(1000000)
> start <- Sys.time()
> y <- sum(x)
> Sys.time() - start
Time difference of 0.001904726 secs
> print(y)
[1] -298.7991
```

```
> start <- Sys.time()
> y <- 0
> for (i in 1:1000000) {
+     y <- y + x[i]
+ }
> Sys.time() - start
Time difference of 0.8191297 secs
> print(y)
[1] -298.7991
```

Notice that the loop is several orders of magnitude slower than `sum`. Functions like `apply` are popular among `R` programmers because they help to avoid loops, but they exhibit the same performance issues because they are essentially doing the same thing as a loop. Many of the core computations, such as matrix algebra, are written in `C` and therefore give excellent performance if you can leverage them.

7.3 Interfacing with C Code

Sometimes it is not possible to write pure `R` code to make efficient use of the core `R` functions, as suggested in Section 7.2. In this case, you may want to consider writing your function in `C` with an `R` interface. This requires some work, but provides an efficient `R` function which can be called like any other `R` function. An example is given in the `OverdispersionModelsInR` package; see the `find.vertices` function which is used in computing the Mixture Link binomial density. The package `Rcpp` is also available to simplify manipulation of `R` objects (e.g. matrices and vectors) in `C/C++`.

7.4 Parallel Computing

Sometimes the performance of a program can be greatly improved by splitting it into pieces which can run simultaneously. This is the approach taken in parallel programming. Modern computers often have multiple processor cores. Some institutions host large scale parallel computers with hundreds or thousands of processor cores. The authors' institution hosts such a machine, through the High Performance Computing Facility (HPCF, `www.umbc.edu/hpcf`).

`R` supports parallel programming through packages developed by the community. The `snow` package provides a master/worker paradigm through constructs such as a parallel version of `apply`, where the master doles out tasks to workers. The `pbdR` package provides an `MPI` paradigm where parallel processes are peers and communicate with each other to carry out a job. The `RHIPE` package provides a map-reduce (Hadoop) paradigm, which has become extremely popular for working on very large distributed datasets. More options for high performance and parallel computing are discussed at `cran.r-project.org/web/views/HighPerformanceComputing.html`.

Some information about getting started with parallel `R` (especially `snow` and `pbdR`) on the HPCF computing cluster is provided at www.umbc.edu/hpcf $\Longrightarrow$ "Resources for HPCF Users" $\Longrightarrow$ "R".

Also see the tutorial (Raim, 2013) on pbdR. These materials can serve as illustrative examples of R on a high performance computing cluster; however, some procedures are specific to the HPCF environment and may be different for your cluster.

7.5 Making a Package

R users can create packages containing code and data. This provides a convenient method for sharing your work with the general R community, a group of colleagues, or just to organize your own code for yourself. In this document, we have walked through some of the development of the OverdispersionModelsInR package. To learn how to create your own package, a good starting place is (Leisch, 2008).

Acknowledgements

This document began as an R supplement to the workshop "Analysis of Overdispersed Data using SAS", presented at the 8th Annual Probability and Statistics Day at UMBC on April 18, 2014. We are grateful to conference organizers and attendees for the opportunity to present this material. The first author graciously thanks Dr. Matthias Gobbert and the UMBC High Performance Computing Facility (www.umbc.edu/hpcf) for financial support as an RA.

References

Friedrich Leisch. Creating R packages: A tutorial. In Paula Brito, editor, *Compstat 2008-Proceedings in Computational Statistics*, Heidelberg, Germany, 2008. Physica Verlag. URL http://cran.r-project.org/doc/contrib/Leisch-CreatingPackages.pdf.

Jorge G. Morel and Nagaraj K. Neerchal. *Overdispersion Models in SAS*. SAS Institute, 2012.

Nagaraj K. Neerchal and Jorge G. Morel. Large cluster results for two parametric multinomial extra variation models. *Journal of the American Statistical Association*, 93(443):1078–1087, 1998.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2014. URL http://www.r-project.org.

Andrew M. Raim. Introduction to distributed computing with pbdR at the UMBC High Performance Computing Facility. Technical Report HPCF–2013–2, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2013. URL http://www.umbc.edu/hpcf/publications/index.html.

Andrew M. Raim. *Computational Methods in Finite Mixtures using Approximate Information and Regression Linked to the Mixture Mean*. PhD thesis, Department of Mathematics and Statistics, University of Maryland, Baltimore County, 2014.

Andrew M. Raim, Minglei Liu, Nagaraj K. Neerchal, and Jorge G. Morel. On the method of approximate Fisher scoring for finite mixtures of multinomials. *Statistical Methodology*, 18: 115–130, 2014.

Santosh C. Sutradhar, Nagaraj K. Neerchal, and Jorge G. Morel. A goodness-of-fit test for overdispersed binomial (or multinomial) models. *Journal of Statistical Planning and Inference*, 138(5):1459–1471, 2008.