# `exactLoglinTest`: A Program for Monte Carlo Conditional Analysis of Log-linear Models

Brian S. Caffo

June 30, 2011

Note, a more detailed, though static, description of the software is available in [3]. To reference `exactLoglinTest`, please use:

```
title={Exact Hypothesis Tests for Log-linear Models with {exactLoglinTest}},
author={Caffo, B},
journal={Journal of Statistical Software},
volume={17},
number={7},
pages={1--17},
year={2006}}
```

## 1 Introduction

Nuisance parameters are parameters that are not of direct interest to the inferential question in hand. In a frequentist or likelihood paradigm, a common tool for eliminating nuisance parameters is to condition on their sufficient statistics. The same technique is useful (though rarely used) in a Bayesian settings, as it eliminates the need to put priors on nuisance parameters.

For log-linear models, conditional analysis suffers from two main drawbacks.

1. The set of lattice points contained in the conditional distribution is difficult to manage, computationally or analytically.
2. The sufficient statistics for the nuisance parameters are not ancillary to the parameters of interest.

In this manuscript we address only the first drawback using `exactLoglinTest`.

## 2 The Problem

The observed data, $y = (y_1, \ldots, y_n)$, are modeled as Poisson counts with a means, $\mu = (\mu_1, \ldots, \mu_n)$, satisfying

$$\log \mu = x\beta$$

under the null hypothesis. Here $x$ is a full rank $n \times p$ design matrix. It is easily shown that the sufficient statistics for $\beta$ under the null hypothesis are $x^t y$, where a superscript $t$ denotes a transpose. Let $h$ be a test statistic of interest where larger values of $h$ support the alternative hypothesis. Two examples are the Pearson Chi-Squared statistic and the deviance. An exact test relative to $h$ can be performed via the conditional P-value

$$\text{Prob}\{h(y) \geq h(y_{obs}) | x^t y = x^t y_{obs}\} = \sum_{\{y \in \Gamma\}} \frac{I\{h(y) \geq h(y_{obs})\}}{C \prod y_i!}$$

where $y_{obs}$ is the observed table, $C$ is a normalizing constant and $\Gamma = \{y | x^t y = x^t y_{obs}\}$ (often referred to as the reference set).

The term "exact" is used to refer to tests that guarantee the nominal type I error rate unconditionally. Thus a test that never rejects the null hypothesis is technically exact in any situation. Therefore, exactness is not in itself a sufficient condition for a test to be acceptable. Moreover, this example (never

rejecting) is particularly relevant in our setting because $\Gamma$ may contain one or few elements. Hence the conditional P-value will be exactly or near one regardless of the evidence in the data vis-a-vis the two hypotheses. However, it is also the case that the conservative conditional tests can produce P-values that are smaller than those calculated via Chi-squared approximations (see Subsection 4.2 for an example).

# 3 exactLoglinTest

The software `exactLoglinTest` is an implementation of the algorithms presented in [2] and [4]. At the heart of both algorithms is a sequentially generated rounded normal approximation to the conditional distribution. We refer the reader to those papers for a more complete description.

You can obtain a copy of `exactLoglinTest` at as well as a no-web [7] version of this document at

`http://www.biostat.jhsph.edu/~bcaffo/downloads.htm`

You can install `exactLoglinTest` with `R CMD INSTALL`, on Unix and Linux, while the binaries are available for Windows. Assuming it is installed, one can load `mcexact` with

```
> library(exactLoglinTest)
> set.seed(1)
```

Here, the optional argument `lib.loc` is necessary if the package has been installed into one of the paths that R automatically checks. We also set the random number seed to a specific value which is a good practice for Monte Carlo procedures.

# 4 Examples

## 4.1 Residency Data

Assuming `exactLoglinTest` has been properly installed, the residency data can be obtained by the command

```
> data(residence.dat)
```

This data is a 4×4 table of persons' residence in 1985 by their residence in 1980. See Table 1 for the complete data. The data frame, `residence.dat`, contains the counts stacked by the rows. The extra term `sym.pair` is used to fit a quasi-symmetry model. For details on the quasi-symmetry model see [1]. To obtain a Monte Carlo goodness of fit test of quasi-symmetry versus a saturated model involves the following command

```
> resid.mcx <- mcexact(y ~ res.1985 + res.1980 + factor(sym.pair),
+     data = residence.dat, nosim = 10^2, maxiter = 10^4)
> resid.mcx

                deviance    Pearson
observed.stat 2.98596233 2.98198696
pvalue        0.43615976 0.43615976
mcse          0.03240488 0.03240488
```

The default method is the importance sampling of [2]. Using this method, the number of desired simulations `nosim` may not be met in `maxiter` iterations and no warning is issued if this occurs. The returned value is a list storing the results of the Monte Carlo simulation and all of the relevant information necessary to restart the simulation. More information can be obtained with `summary`

```
> summary(resid.mcx)

Number of iterations       =  100
t degrees of freedom       =  3
Number of counts           =  16
df                         =  3
Next update has nosim      =  100
Next update has maxiter    =  10000
```

```
Proportion of valid tables =  1

                deviance    Pearson
observed.stat 2.98596233 2.98198696
pvalue        0.43615976 0.43615976
mcse          0.03240488 0.03240488
```

The $t$ degrees of freedom refers to degrees of freedom used as a tuning parameter within the algorithm while the `df` refers to the model degrees of freedom. In this case, the Monte Carlo standard error, `mcse`, seems too large. As mentioned previously, `mcexact` , stores the relevant information for restarting the simulation

```
> resid.mcx <- update(resid.mcx, nosim = 10^4, maxiter = 10^6)
> resid.mcx

                 deviance     Pearson
observed.stat 2.985962330 2.981986964
pvalue        0.400636040 0.400930887
mcse          0.003196472 0.003196835
```

It is important to note that `update` only resumes the simulation with changes to simulation-specific parameters. It will not allow users to change the model formulation; one must rerun `mcexact` independently to do that.

This example illustrates the point that the underlying algorithms are very efficient when the cell counts are large. Of course, when this is the case, the large sample approximations are nearly identical to the conditional results

```
> pchisq(c(2.986, 2.982), 3, lower.tail = FALSE)

[1] 0.3937887 0.3944088
```

## 4.2 Pathologists' Tumor Ratings

The following example is interesting in that the large sample results differ drastically from the conditional results. Moreover, the conditional results are less conservative. The data, given in Table 2 can be obtained via

```
> data(pathologist.dat)
```

A uniform association model accounts for the ordinal nature of the ratings by associating ordinal scores with the pathologist's ratings [see 1]. Specifically, we can test a uniform association model against the saturated model with

```
> path.mcx <- mcexact(y ~ factor(A) + factor(B) + I(A * B), data = pathologist.dat,
+     nosim = 10^4, maxiter = 10^4)
> summary(path.mcx)

Number of iterations       =  0
t degrees of freedom       =  3
Number of counts           =  25
df                         =  15
Next update has nosim      =  10000
Next update has maxiter    =  10000
Proportion of valid tables =  0

              deviance  Pearson
observed.stat 16.21453 14.72928
pvalue              NaN      NaN
mcse                NaN      NaN
```

The previous code chunk takes about 1 minute on my laptop. It is worth comparing these results to the asymptotic Chi-squared results

```
> pchisq(c(16.214, 14.729), 15, lower.tail = FALSE)

[1] 0.3679734 0.4711083
```

### 4.3 Alligator Food Choice Data Using MCMC

In this example we illustrate the algorithm from[4] using the data and Poisson log-linear model from Table 3. The alligator data is a good choice for MCMC as the percent of valid tables generated using `method = "bab"` is very small, less than 1% of the tables simulated. It is often the case that the MCMC algorithm will be preferable when the table is large and/or sparse. Of course, using MCMC introduces further complications in reliably running and using the output of the algorithm.

The algorithm from [4] uses local moves to reduce the number of tables with negative entries that the chain produces. You can specify this method by using `method = "cab"`. The parameter `p` represents the average proportion of table entries left fixed. So a chain with `p=.9` will leave most of the table entries fixed from one iteration to the next. A high value of `p` will result in a high proportion of valid (non-negative) simulated tables. Too large of a value of `p` causes the chain to mix slowly because the tables will be very similar from one iteration to the next. However, it is sometimes the case that a small value of `p` will produce too many tables with negative entries. Hence the Metropolis/Hastings/Green algorithm will stay at the current table for long periods and again result in a slowly mixing chain. It is also worth mentioning that for large values of `p` the algorithm is theoretically irreducible, but may not be practically irreducible. Therefore, it is advisable to both tinker with the chain some and make final runs using multiple values of `p`.

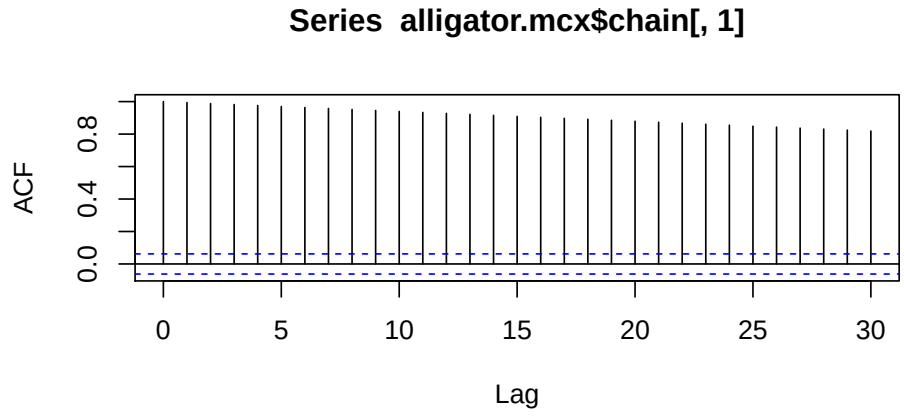
The program allows for the option to save the chain goodness of fit statistics, so that some initial tinkering can be performed. This is specified with the `savechain = TRUE` option. If using impartance sampling, `method = "bab"`, then `savechain` saves both the statistic values and the importance weights on the log scale.

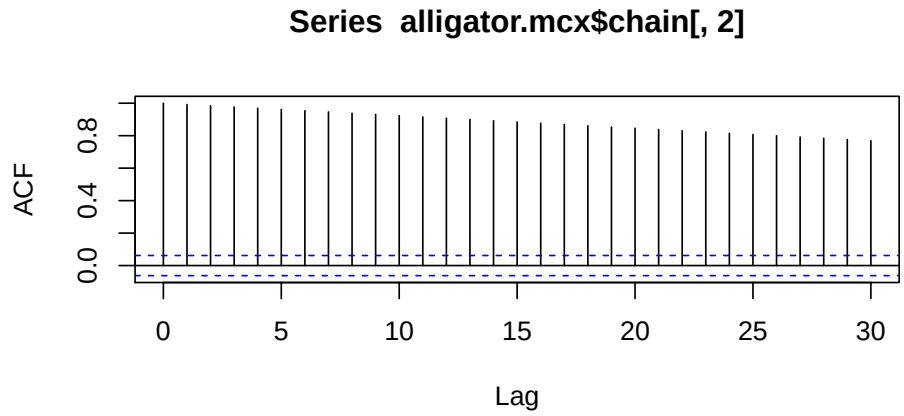
```
> data(alligator.dat)
> alligator.mcx <- mcexact(y ~ (lake + gender + size) * food +
+     lake * gender * size, data = alligator.dat, nosim = 10^3,
+     method = "cab", savechain = TRUE, batchsize = 100, p = 0.4)
> summary(alligator.mcx)

Number of iterations       =  1000
t degrees of freedom       =  3
Number of counts           =  80
df                         =  40
Number of batches          =  10
Batchsize                  =  100
Next update has nosim      =  1000
Proportion of valid tables =  0.02

                deviance    Pearson
observed.stat 50.2636886 52.5676870
pvalue         0.2510000  0.2510000
mcse           0.1217247  0.1217247
```

The chain of goodness of fit statistics are saved in `alligator.mcx$chain`. The saved chain is discarded if the simulations are resumed with `update`, even if `savechain = T` when the simulation is resumed.

We would want to look at the autocorrelation function of the goodness of fit statistics.

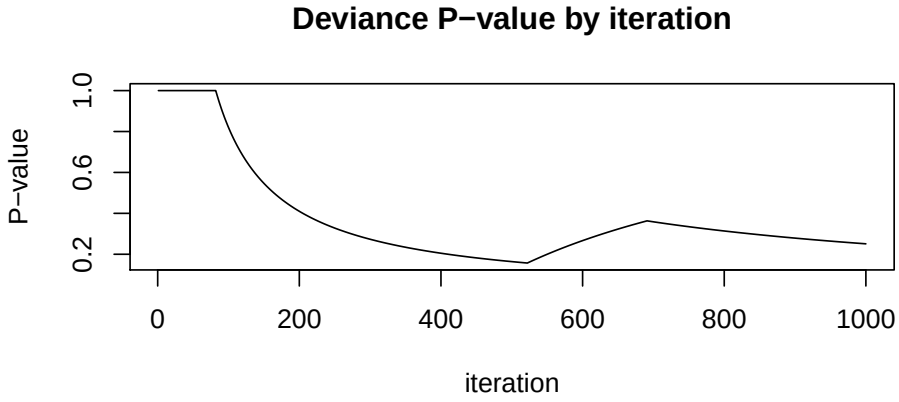
```
> par(mfrow = c(2, 1))
> acf(alligator.mcx$chain[, 1])
> acf(alligator.mcx$chain[, 2])
```

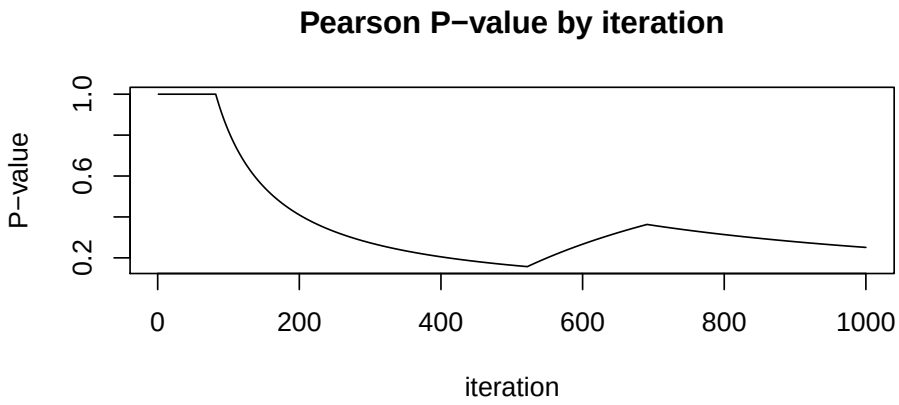



We would also want to look at the chain of P-values.

```
> dev.p <- cumsum(alligator.mcx$chain[, 1] >= alligator.mcx$dobs[1])/(1:alligator.mcx$nosim)
> pearson.p <- cumsum(alligator.mcx$chain[, 1] >= alligator.mcx$dobs[1])/(1:alligator.mcx$nosim)
> par(mfrow = c(2, 1))
> plot(dev.p, type = "l", ylab = "P-value", xlab = "iteration")
> title("Deviance P-value by iteration")
> plot(pearson.p, type = "l", ylab = "P-value", xlab = "iteration")
> title("Pearson P-value by iteration")
```




The P-values have apparently not stabilized. Also, there is an extremely slow decay in the autocorrelations of the chain of goodness of fit statistics. Therefore, we should execute a longer run using large batch sizes. While on the subject of batch sizes, note that `mcexact` does not require the total number of simulations to be a multiple of the batch size. If the algorithm terminates in the middle of completing a batch, it is not used in the P-value calculations. However, the simulations are not wasted if the algorithm is resumed with `update`.

One large final run of this data discarding all of the initial tinkering could be performed by setting `flush = TRUE` as an argument to `update`. Here, `flush = TRUE`, tells `update` to throw out all of the data used in the initial tinkering, except that it starts the new chain from the final table from the initial runs. This is a harmless way to burn the chain in while you are tinkering with it. Of course, the chain can be restarted at the default starting value, the observed data, by simply rerunning `mcexact` .

# 5 Application to Disclosure Limitation

Though there are certainly more rigorous procedures available [see 5], `exactLoglinTest` is a useful tool for exploring disclosure limitation in contingency tables. Consider the Czech Auto Worker's data given in Table 4. Suppose a researcher is concerned about the potential disclosure risk of releasing all two-way marginals from this table. The following code will load the Czech auto worker data into a data frame:

```
> data(czech.dat)
```

We will explore disclosure limitation by simulating tables from the hypergeometric distribution obtained by conditioning on all two way margins. However, we would like to save all of the simulated table entries, not just the deviance and Pearson statistics. This could be accomplished by changing the argument `stat` of `mcexact` to an appropriate statistic. However, the function `simulateConditional` performs this simulation for us. It returns the simulated tables in a matrix with each row being a complete simulated table.

Now we run the chain. Notice the `stat = cell.stat` option to load the newly defined statistic.

```
> chain <- simulateConditional(y ~ (A + B + C + D + E + F)^2, data = czech.dat,
+     method = "cab", nosim = 10^3, p = 0.4)
```

Now, `chain` is a matrix where each row is a simulated table. We were particularly concerned with cells 39, 48, and 55 which contained only one, two and two individuals respectively. Consider the proportion of tables which have greater than 0 but fewer than three individuals

```
> mean(chain[, 39] > 0 & chain[, 39] < 3)

[1] 0.389

> mean(chain[, 48] > 0 & chain[, 58] < 3)

[1] 0.11

> mean(chain[, 55] > 0 & chain[, 55] < 3)

[1] 0.896
```

We used the model in question because this model fixes all two-way margins. However, that model need not fit the data well (in fact, it doesn't). Therefore, in addition to simulating from the hypergeometric density, a user would likely also want to simulate from other densities, such as a uniform distribution on tables with these margins. Though the normal approximations for `exactLoglinTest` were tailored specifically to the hypergeometric density, it allows for other target distributions. Here the density must be specified on the log scale up to a constant. Since a uniform density is simply a constant we use a density that always returns 0.

```
> chain2 <- simulateConditional(y ~ (A + B + C + D + E + F)^2,
+     data = czech.dat, method = "cab", nosim = 10^3, p = 0.4,
+     dens = function(y) 0)
> mean(chain2[, 39] > 0 & chain2[, 39] < 3)

[1] 0.363

> mean(chain2[, 48] > 0 & chain2[, 58] < 3)

[1] 0.59

> mean(chain2[, 55] > 0 & chain2[, 55] < 3)

[1] 1
```

Both simulations suggest that there are plenty of tables with higher counts than the observed counts for cells 39, 48 and 55. Hence the disclosure risk in releasing the two-way marginals seems minimal. However, it should be reiterated that this example is given only to illustrate how to obtain simulated tables from `exactLoglinTest`, further investigation of the chain and the data would be necessary for a thorough analysis of the disclosure risk.

# 6 Discussion

In this manual we investigated three straightforward examples of `exactLoglinTest` and considered a useful extension of the program to disclorsure limitation. The program was initially constructed calculate P-values for goodness of fit tests for contingency tables.

In previous versions of this vignette, we discussed using `exactLoglinTest` for performing tests in exact binomial logistic regression. It is true that conditional Poisson loglinear models contain conditional logistic regression models as special cases. However, we have found that in some cases, the matrices used by `exactLoglinTest` for this purpose are ill conditioned. Thus we no longer recommend `exactLoglinTest` for pursuing conditional logistic regression models, unless the user is particularly familiar with the area. The paper [3] gives more details on exact binomial testing.

# References


[1] Alan Agresti. *Categorical Data Analysis*. Wiley, New York, 1990.

[2] J.G. Booth and R.W. Butler. An importance sampling algorithm for exact conditional test in log-linear models. *Biometrika*, 86:321–332, 1999.

[3] B˜Caffo. Exact hypothesis tests for log-linear models with exactLoglinTest. *Journal of Statistical Software*, 17(7):1–17, 2006.

[4] Brian˜S. Caffo and James˜G. Booth. A markov chain monte carlo algorithm for approximating exact conditional probabilities. *the Journal of Compuatational and Graphical Statistics*, 10:730–745, 2001.

[5] Adrian Dobra, Claudia Tebaldi, and Mike West. Reconstruction of contingency tables with missing data. Technical report, Duke University, 2002.

[6] D.˜E. Edwards and T.˜Havranek. A fast procedure for model search in multidimesional contingency tables. *Biometrika*, 72:339–351, 1985.

[7] Friedrich Leisch. *Sweave User Manual*.


# A Tables

| Residence | Residence in 1985 | | | |
|---|---|---|---|---|
| in 1980 | Northeast | Midwest | South | West |
| Northeast | 11,607 | 100 | 366 | 124 |
| Midwest | 87 | 13,677 | 515 | 302 |
| South | 172 | 225 | 17,819 | 270 |
| West | 63 | 176 | 286 | 10,192 |

Source [1]

Table 1: Residency Data

| | Pathologist B | | | | |
|---|---|---|---|---|---|
| Pathologist A | 1 | 2 | 3 | 4 | 5 |
| 1 | 22 | 2 | 2 | 0 | 0 |
| 2 | 5 | 7 | 14 | 0 | 0 |
| 3 | 0 | 2 | 36 | 0 | 0 |
| 4 | 0 | 1 | 14 | 7 | 0 |
| 5 | 0 | 0 | 3 | 0 | 3 |

Source [1]

Table 2: Pathologist Agreement Data

| | | | Primary Food Choice | | | | |
|---|---|---|---|---|---|---|---|
| Lake | Gender | Size | Fish | Invert | Reptile | Bird | Other |
| 1 | Male | Small | 7 | 1 | 0 | 0 | 5 |
| | Male | Large | 4 | 0 | 0 | 1 | 2 |
| | Female | Small | 16 | 3 | 2 | 2 | 3 |
| | Female | Large | 3 | 0 | 1 | 2 | 3 |
| 2 | Male | Small | 2 | 2 | 0 | 0 | 1 |
| | Male | Large | 13 | 7 | 6 | 0 | 0 |
| | Female | Small | 3 | 9 | 1 | 0 | 2 |
| | Female | Large | 0 | 1 | 0 | 1 | 0 |
| 3 | Male | Small | 3 | 7 | 1 | 0 | 1 |
| | Male | Large | 8 | 6 | 6 | 3 | 5 |
| | Female | Small | 2 | 4 | 1 | 1 | 4 |
| | Female | Large | 0 | 1 | 0 | 0 | 0 |
| 4 | Male | Small | 13 | 10 | 0 | 2 | 2 |
| | Male | Large | 9 | 0 | 0 | 1 | 2 |
| | Female | Small | 3 | 9 | 1 | 0 | 1 |
| | Female | Large | 8 | 1 | 0 | 0 | 1 |

Source [1]
Model (FG, FL, FS, LGS) where F=food choice, L=lake, S=size, G=gender.

Table 3: Alligator Data

| | | | | B | no | | yes | |
|---|---|---|---|---|---|---|---|---|
| F | E | D | C | A | no | yes | no | yes |
| neg | small | small | no | | 44 | 40 | 112 | 67 |
| | | | yes | | 129 | 145 | 12 | 23 |
| | | large | no | | 35 | 12 | 80 | 33 |
| | | | yes | | 109 | 67 | 7 | 9 |
| | large | small | no | | 23 | 32 | 70 | 66 |
| | | | yes | | 50 | 80 | 7 | 13 |
| | | large | no | | 24 | 25 | 73 | 57 |
| | | | yes | | 51 | 63 | 7 | 16 |
| pos | small | small | no | | 5 | 7 | 21 | 9 |
| | | | yes | | 9 | 17 | 1 | 4 |
| | | large | no | | 4 | 3 | 11 | 8 |
| | | | yes | | 14 | 17 | 5 | 2 |
| | large | small | no | | 7 | 3 | 14 | 14 |
| | | | yes | | 9 | 16 | 2 | 3 |
| | | large | no | | 4 | 0 | 13 | 11 |
| | | | yes | | 5 | 14 | 4 | 4 |

Source [5] originally appeared in [6].

Table 4: Czech Auto Workers Data

| | | | Class | | | |
|---|---|---|---|---|---|---|
| Surv | Sex | Age | Crew | First | Second | Third |
| no | F | Child | 0 | 0 | 0 | 17 |
| | | Adult | 3 | 4 | 13 | 89 |
| | M | Child | 0 | 0 | 0 | 35 |
| | | Adult | 670 | 118 | 154 | 387 |
| yes | F | Child | 0 | 1 | 13 | 14 |
| | | Adult | 20 | 140 | 80 | 76 |
| | M | Child | 0 | 5 | 11 | 13 |
| | | Adult | 192 | 57 | 14 | 75 |