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

Brian S. Caffo

July 28, 2003

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`.

1 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 β 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 Γ 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 3.2 for an example).

2 exactLoglinTest

The software `exactLoglinTest` is an implementation of the algorithms presented in [2] and [3]. 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 more detailed no-web [4] 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 the program with.

```
R> library(exactLoglinTest)
R> 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.

3 Examples

3.1 Residency Data

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

```
R> 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

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

               deviance   Pearson
observed.stat 2.9859623 2.9819870
pvalue        0.4103531 0.4103531
mcse          0.2077671 0.2077671
```

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`

```
R> 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 2.9859623 2.9819870
pvalue        0.4103531 0.4103531
mcse          0.2077671 0.2077671
```

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

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

                deviance    Pearson
observed.stat 2.98596233 2.98198696
pvalue        0.39875302 0.39820666
mcse          0.01994486 0.01993785
```

It is important to note that `update` only resumes the simulation with changes to some 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

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

[1] 0.3937887 0.3944088
```

3.2 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

```
R> 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

```
R> path.mcx <- mcexact(y ~ factor(A) + factor(B) + I(A * B), data = pathologist.dat,
+      nosim = 10^5, maxiter = 10^6)
R> summary(path.mcx)

Number of iterations       =  1e+05
T degrees of freedom       =  3
Number of counts           =  25
df                         =  15
Next update has nosim      =  1e+05
Next update has maxiter    =  1e+06
Proportion of valid tables =  1
```

```
                  deviance      Pearson
observed.stat 16.214350396 14.729165468
pvalue         0.044960499  0.134389180
mcse           0.001848729  0.002837652
```

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

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

[1] 0.3679734 0.4711083
```

3.3 Alligator Food Choice Data Using MCMC

In this example we illustrate the algorithm from[3] 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 [3] 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`.

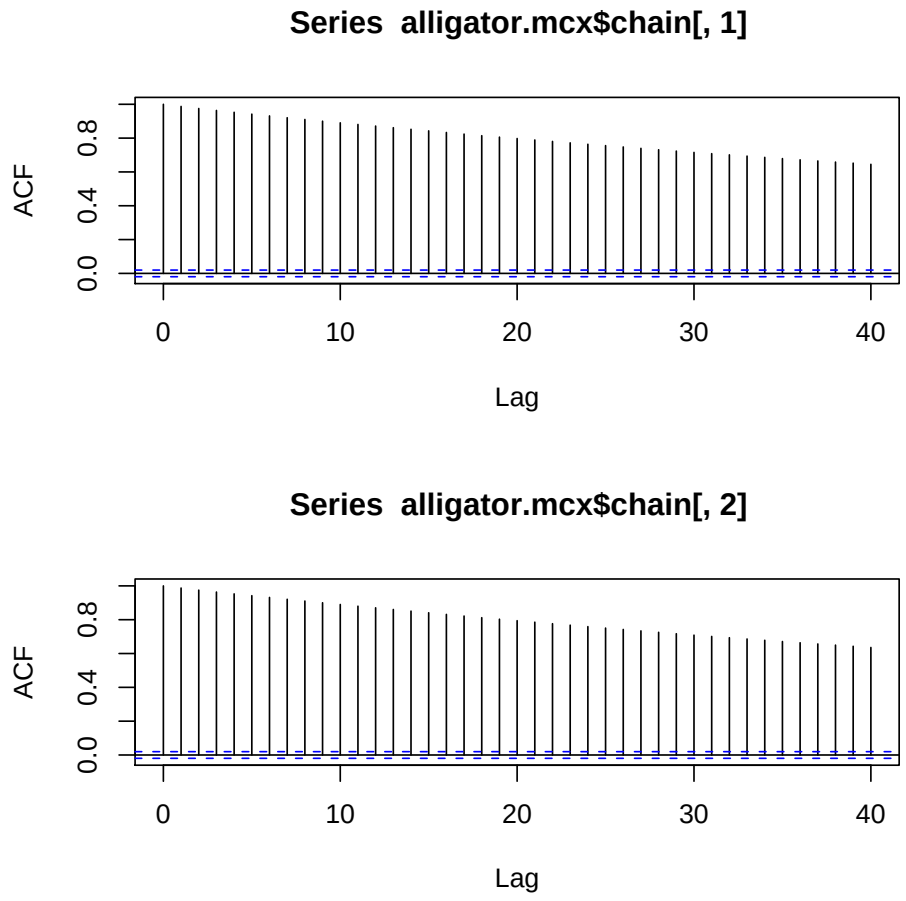
The program allows for the option to save the chain goodness of fit statistics, so that some initial tinkering can be performed. This is specified with the `savechain = TRUE` option. If using impartance sampling, `method = "bab"`, then `savechain` saves both the statistic values and the importance weights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

```
R> data(alligator.dat)
R> alligator.mcx <- mcexact(y ~ (lake + gender + size) * food +
+      lake * gender * size, data = alligator.dat, nosim = 10^4,
+      method = "cab", savechain = TRUE, batchsize = 100, p = 0.4)
R> summary(alligator.mcx)

Number of iterations       =  10000
T degrees of freedom       =  3
Number of counts           =  80
df                         =  40
Number of batches          =  100
Batchsize                  =  100
Next update has nosim      =  10000
Proportion of valid tables =  0.0492

                  deviance     Pearson
observed.stat 50.26355592 52.56691167
pvalue         0.22370000  0.12310000
mcse           0.03334866  0.02793482
```

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.

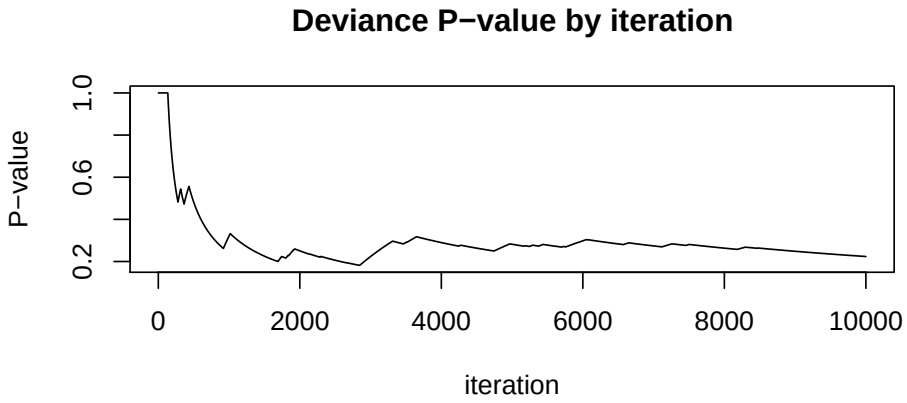
We would want to look at the autocorrelation function of the goodness of fit statistics.

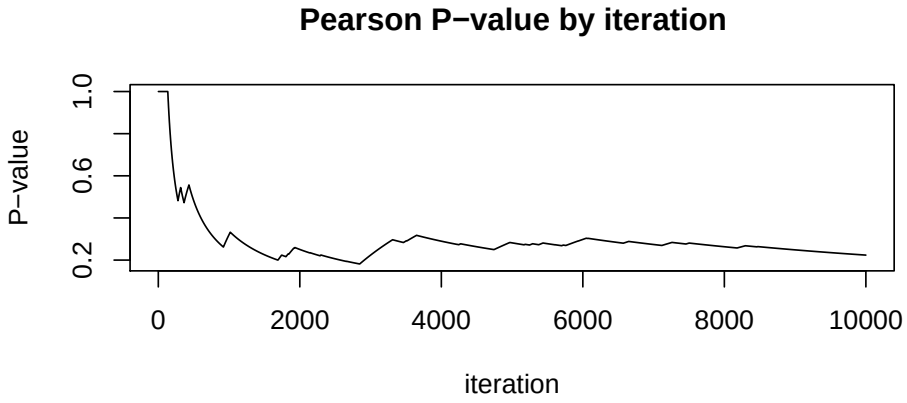
```
R> library(ts)
R> par(mfrow = c(2, 1))
R> acf(alligator.mcx$chain[, 1])
R> acf(alligator.mcx$chain[, 2])
```



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

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





Though the P-values have apparently stabilized and are clearly larger than most normal type I error rates, 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 could be performed using `update` again. The option, `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` .

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] 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.

Residence in 1980	Residence in 1985			
	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

Table 1: Residency Data

Source [1]

	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

Table 2: Pathologist Agreement Data

Source [1]

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

A Tables

			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

Table 3: Alligator Data

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