maRkov: Goodness-of-fit Tests for Binary Markov Chains

By
Conrad Cartmell

A THESIS

submitted to

Oregon State University

University Honors College

in partial fulfillment of
the requirements for the
degree of

Honors Baccalaureate of Science in Mathematics
(Honors Scholar)

Presented May 24, 2016
Commencement June 2016

AN ABSTRACT OF THE THESIS OF

Conrad Cartmell for the degree of Honors Baccalaureate of Science in Mathematics presented on May 24, 2016. Title:
maRkov: Goodness-of-fit Tests for Binary Markov Chains

Abstract approved:

Debashis Mondal

Markov chains, discovered over one hundred years ago by Andrey Markov in an attempt to analyze the distribution of vowels and consonants in Russian poetry, have evolved in multiple fields over the last century to become a mainstay of statistical modeling and computation. maRkov is a package of software to be deployed within the R programming environment that allows for various tests to be performed to check the appropriateness of different Markov Chain models for sets of observed binary data.

Key Words: Markov chains, LRT, binary data, backward forward algorithm, exchangeable sequence

Corresponding e-mail address: cartmelc@oregonstate.edu

maRkov: Goodness-of-fit Tests for Binary Markov Chains

By
Conrad Cartmell

A THESIS

submitted to

Oregon State University

University Honors College

in partial fulfillment of
the requirements for the
degree of

Honors Baccalaureate of Science in Mathematics
(Honors Scholar)

Presented May 24, 2016
Commencement June 2016

Honors Baccalaureate of Science in Mathematics project of Conrad Cartmell presented on May 24, 2016

APPROVED:

Debashis Mondal, Mentor, representing Statistics

Charlotte Wickham, Committee Member, representing Statistics

Mina Ossiander, Committee Member, representing Mathematics

Toni Doolen, Dean, Oregon State University Honors College

I understand that my project will become part of the permanent collection of Oregon State University Honors College. My signature below authorizes release of my project to any reader upon request.

Conrad Cartmell, Author

Contents

1 Introduction

The project that makes up the backbone of this thesis is an R package called maRkov. R is an open source statistical programming environment based on the S programming language, which is maintained and developed by the R Core Team. The functionality of R is supported by by what are called packages, modules of software created largely by people unaffiliated with the R Core Team. These packages, for the most part, are what makes R so useful for most people. They give users in a variety of fields tools that are formulated specifically for them, thereby reducing the required degree of programming knowledge required to perform complex analysis, as well as giving them tools to prepare graphics and documentation of those results.

The maRkov package gives users the ability to answer two questions about binary Markov chains.

- Is a sequence of binary data adequately described adequately by a first order Markov chain, when compared to the alternative of a second order one?
- Is a set of multiple sequences of binary data described adequately by a single Markov chain, when compared to the alternative of multiple maRkov chains?

These two questions are answered by the single_binary_test() and multiple_binary_test() functions, respectively, in the maRkov package. What follows in this thesis is the documentation for said package, from an overview of every included function to the various included vignettes that give extended information on the relevant test statistics and practical examples of the package in action.

2 Vignettes

2.1 Explanation of Test Statistics

See next page...

Explanation of Test Statistics

Conrad Cartmell

2016-05-28

The test statistics utilized in the `maRkov` package are drawn from the paper "Exact Goodness-of-Fit Tests for Markov Chains" (Besag and Mondal 2013). This vignette will attempt to give a brief introduction so that the user of the `maRkov` package is better equipped to understand the guts of the included programs.

Introduction to Markov chains

Markov chains were developed over a hundred years ago by Andrey Markov in an attempt to analyze the distribution of vowels and consonants in Russian poetry. Over the years, their use has expanded into a wide array of sciences, including genetics, biology, and physics.

Despite being extremely powerful tools, Markov chains are very simple processes at their most basic level. The chains are composed of a set of states, called a state space, as well as a set of probabilities for transitioning from each state to each other state, usually encoded in a transition matrix. Because this package is primarily concerned with binary (two state) Markov chains, we'll be using those as examples.

The defining feature of Markov chains is that they are memoryless. That is to say, the probability of transitioning from one state to another is not affected in any way by the history of the chain. The only determining factor for the transition probabilities is the state that they are currently in.

State	0	1
0	.6	.4
1	.3	.7

The above table is a transition matrix. The first column represents the state that the Markov chain is currently in, and the first row represents the state that it is transitioning to. In this case, the probability of transitioning from state 0 to 0 is .6, from 0 to 1 is .4, from 1 to 0 is .3, etc...

The Markov chain described in the above table is a first order Markov chain. This means that each state consists of the state of the process in a single time step. But it doesn't have to be this way. We can expand our state space so that it includes all possible doubles of individual states, thus capturing two time steps at once. An example of a second order Markov chain is given in the table below.

State	(0,0)	(0,1)	(1,0)	(1,1)
(0,0)	.6	.4	0	0
(0,1)	0	0	.3	.7
(1,0)	.6	.4	0	0
(1,1)	0	0	.3	.7

This table is read in the same way as the first. The first thing that pops out is the fact that it is impossible to transition to a certain states from any given states. This is because, despite the fact that each state captures two time steps, it is still only possible to take one step at once. Therefor, if the process is in, say, state (*0*,**0**), when time moves forward one step, the *italic 0* in the first position will be "pushed out" of the state, and the **bold 0** in the second position will be pushed into its place. Hence, the process is in state (0,0), the next state it is in will necessarily have a first element of 0.

While second order Markov chains are certainly useful, and may in many cases have more power to describe

real life processes, there is a serious issue of computational efficiency. As you can see, increasing the order of a Markov chain increases the number of states by a factor of the number of states in the first order chain. Thus, to reduce computational demand, it is well within our interest to try to keep the total number of states at a minimum.

Questions of interest & test statistics

The general principle underpinning the package is that it is possible to use single or multiple real sequences of binary data to produce a set of possible exchangeable sequences that are generated to preserve a certain property found in the original sequence This property is the count of transitions between states of the chain. For example, the sequence of data `a = { 1, 0, 0, 0, 1, 0, 1}` has two transitions from 1 to 0, two transitions from 0 to 0, and two transitions from 0 to 1. An example of a different sequence with the same transition counts would be `b = { 1, 0, 0, 1, 0, 0, 1}`.

In order to generate a large number of such sequences that are independent of the original one, the the procedure attempts swapping entries in a supplied sequence some number of times. For each attempted swap, the sequence is checked to make sure that the transition counts are the same. If they are, the swap is made and the sequence is saved. If not, another swap is attempted. The swaps that are attempted are determined by randomly generating indices of the sequence(s).

Once all the swaps have been attempted, the resulting sequence of data is saved. Then, using that new sequence as a starting point for each new one, the procedure attempts *more* swaps, after which it saves the resulting sequence. However many sequences of generated data are desired are created in this fashion, and all are independent from the original supplied sequence.

Single sequences

For single sequences of data, the primary question of interest addressed by the `maRkov` package is: is a first order binary Markov chain appropriate to describe the given sequence compared to a second order one? The function used to answer this question is `single_binary_test()`, which gives three test statistics: the likelihood ratio test statistic, the Pearson's chi-squared (χ^2) test statistic, and the run test statistic for a run of a given length. The likelihood ratio test statistic addresses the above question of interest. The χ^2 test statistic functions as an approximation of the likelihood ratio test statistic, because they both have similar distributions. Lastly, the run test statistic is useful for judging the homogeneity of the sequence, although it does not test for a rigorous question of interest presented here.

Once `single_binary_test()` has generated a sufficient amount of exchangeable sequences, it goes about generating the aforementioned test statistics. Since the test statistics in the `maRkov` package are concerned with determining the appropriateness of first order chains against second order ones, a table of second order transition counts is compiled for each sequence. From this table, the program generates test statistics.

$$LRT = 2\sum_{i,j,k} n_{ijk} log\left(\frac{\frac{n_{ijk}}{n_{ij+}}}{\frac{n_{+jk}}{n_{+j+}}}\right)$$

$$\chi^2 = \sum_{i,j,k} \frac{\left(n_{ijk} - \frac{n_{ij+}n_{+jk}}{n_{+j+}}\right)^2}{\frac{n_{ij+}n_{+jk}}{n_{+j+}}}$$

In the equations above, n_{ijk} represents the frequency of the transition from state ij to state jk in a second order chain. The $n's$ with $+$ in the subscript indicate summation across all values of either i, j, k or some combination thereof. It should be noted, additionally, that $i = j = k = \{0, 1\}$.

Run test statistics simply represent the number of times that a run of a certain specified length occurs within the sequence. The exact formula is shown below for a run of length p, where x_i is the i^{th} index of a sequence of data x of length l.

$$Run = \sum_{i=1}^{l-p} 1_{x_i=1 \& x_{i+1}=1 \& \cdots \& x_{i+p}=1}$$

Multiple sequences

For multiple binary chains, the question of interest is different. Now, instead asking whether to use a first or second order chain, we ask ourselves whether a single binary Markov chain is adequate to describe a given set of sequences of data, or if multiple binary Markov chains are required. The likelihood ratio test statistic is used to answer this question. The χ^2 distribution is once again an approximation of the distribution of likelihood ratio test statistics.

Unlike `single_binary_test()`, `multiple_binary_test()` isn't at all concerned with second order chains, so this time a table of first order transitions is compiled.

$$LRT = 2 \sum_{i,j,t} n_{ijk}^{(t)} * log \left(\frac{\frac{n_{ijk}^{(t)}}{n_{ij+}^{(t)}}}{\frac{n_{+jk}^{(t)}}{n_{+j+}^{(t)}}} \right)$$

$$\chi^2 = \sum_{t=0}^{n} \sum_{i,j} \frac{\left(n_{ij}^{(t)} - \frac{n_{i+}^{(t)} n_{ij}^{(+)}}{n_{i+}^{(+)}} \right)^2}{n_{i+}^{t} + \frac{n_{ij}^{(+)}}{n_{i+}^{(+)}}}$$

Where $n_{ij}^{(t)}$ represents the frequency of the transition from state i to state j in the t^{th} chain in the set. $+$ indicates the same thing as in the single chain case. $i = j = \{0, 1\}$, and t is the set of natural numbers smaller than the number of new sets of sequences to generate, which is given as the parameter `n` in `multiple_binary_test()`.

The run test statistic is calculated in the same fashion as the run test statistic in the single sequence case, but the run test statistics from each sequence are added together. The exact formula is shown below for a run of length p, where $x_i^{(j)}$ is the i^{th} index of a sequence of data $x^{(j)}$ within a set of n sequences x, all of length l.

$$Run = \sum_{j=}^{n} \sum_{i=1}^{l-p} 1_{x_i^{(j)}=1 \& x_{i+1}^{(j)}=1 \& \cdots \& x_{i+p}^{(j)}=1}$$

References

Besag, J., and D. Mondal. 2013. "Exact Goodness-of-Fit Tests for Markov Chains." *Biometrics* 69 (2): 488–96. doi:10.1111/biom.12009.

2.2 Snoqualmie Falls Rainfall

See next page...

Snoqualmie Falls Rainfall

Conrad Cartmell

2016-05-29

It rains a lot in Washington. This will come as no surprise to anyone who has been to the Evergreen State, but the specific patterns of the rainfall in the region are important to scientists. For this example, we will be investigating whether precipitation at Snoqualmie Falls Washington during the month of January can be modeled with a single binary Markov chains. To determine the answer to this question, we will be testing the null hypothesis of a simple binary chain against the alternative of a second order binary chain for every year of rainfall data that we have from Snoqualmie Falls, which runs from 1948 to 1983, and then test the null hypothesis that we only need one chain for all of the years against the alternative that we need different chains for different years. The data is sourced from Stochastic Modeling of Scientific Data (Guttorp and Minin 1995).

To do this, we will be using the `single_binary_test()` and `multiple_binar_test()` functions. But first we need to encode the rainfall data in the proper way. The data is below. Note that a "1" denotes that there was more than 0.01 inches of rain on a given day, and a "0" denotes that there was less rainfall than that. Each row is a separate year and each column is a day of the month of January.

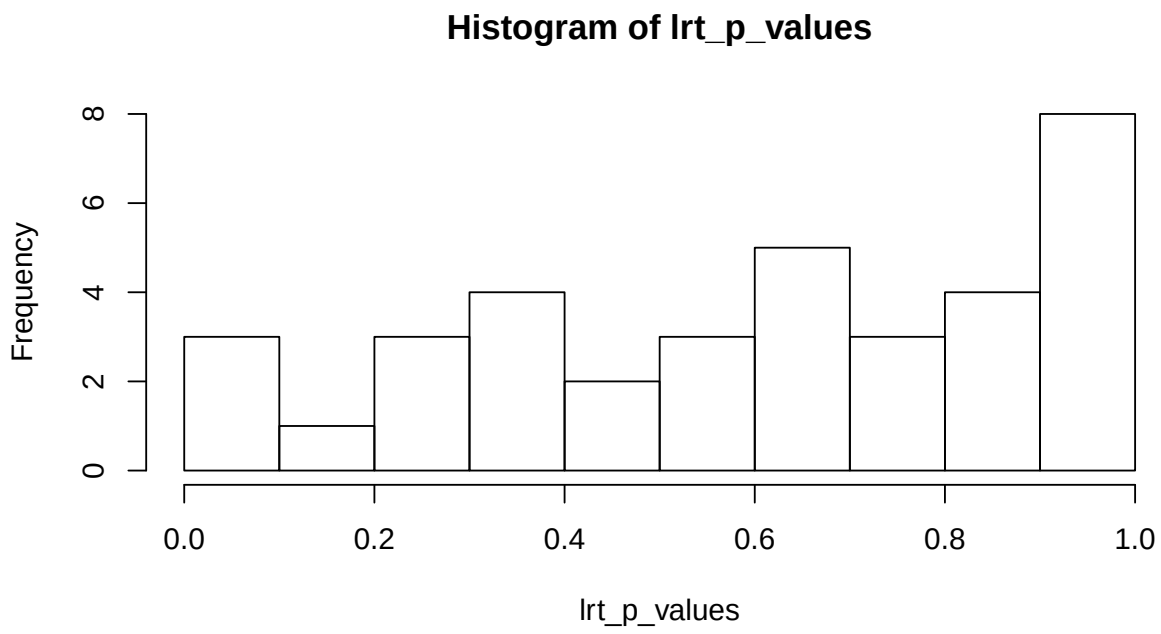
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1948	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	1	0	1	1	1	1	0	0	0	1	1	1
1949	1	1	0	0	1	1	1	1	0	0	0	0	0	1	0	0	1	0	1	0	1	0	0	0	0	1	1	0	0	1	0
1950	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	0	0	0
1951	1	1	1	1	1	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
1952	0	0	1	1	1	1	1	1	1	1	1	1	0	1	0	0	0	1	1	1	1	1	1	1	1	1	1	0	1	1	1
1953	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1954	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1
1955	1	1	0	1	1	0	0	0	0	0	0	1	1	1	1	1	1	0	0	1	0	1	1	1	1	0	0	1	0	1	1
1956	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1	1	0	0
1957	1	1	1	1	0	0	1	1	0	1	1	0	0	1	0	0	1	0	1	1	1	1	1	0	0	0	0	0	0	1	1
1958	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	0	1	1	1	1	1
1959	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
1960	0	1	0	0	1	1	1	1	0	1	1	1	0	0	1	1	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0
1961	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	1	1	0	0	0	1	1	1	1
1962	0	1	1	0	1	1	1	1	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0	1	1	1	1	1	0	0	0
1963	1	1	1	0	1	1	1	1	1	1	0	0	1	1	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	1	1
1964	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	0	1	1	0	1	1	1	1	1
1965	1	1	1	1	1	1	1	1	0	0	0	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
1966	1	1	1	1	1	1	1	1	0	1	1	1	1	1	0	1	1	1	0	0	0	0	1	0	1	0	1	1	1	1	1
1967	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	0	0
1968	1	0	1	1	1	0	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	0	0	1	1
1969	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	0	0	1	0	1	0	1	1	1	1
1970	0	0	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1
1971	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	0
1972	1	1	0	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0
1973	1	1	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	0	0	0	1	1	0
1974	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1975	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	0	0	0	0
1976	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	0	0	1	0	0	0	0	0
1977	1	1	1	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1
1978	0	1	1	1	1	1	1	1	1	1	1	0	1	1	0	1	0	0	1	1	1	1	0	0	1	1	1	1	0	0	1
1979	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	1	0	1	1	1	0	0	1	1	0	1	1	0	0	0	0
1980	1	1	0	1	1	0	1	1	1	1	1	1	1	1	1	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1
1981	0	0	0	0	1	1	0	1	1	0	0	0	0	0	0	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	0
1982	1	1	1	1	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1
1983	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	0	1	0	0

We'll need to encode this data in a way that `single_binary_test` and `multiple_binary_test()` can understand, so we're going to turn it into a matrix with rows representing each single year chain, and columns representing days of January. Luckily, the as.matrix function makes it easy to do this.

```
snoqualmie_matrix <- as.matrix(maRkov::snoqualmie)
```

Before we can examine whether a simple binary Markov chain is appropriate for describing rainfall in the month of January in general, we need to ask ourselves if each January that we have is appropriately modeled by a simple binary Markov chain. To determine this, we will generate a vector of likelihood ratio p-values for each year, and a histogram:

```
lrt_p_values <- apply(snoqualmie_matrix, 1,
                      function(X)(single_binary_test(binary_chain = X,
                                                     swaps = 10000, n = 10000))$p_value_lrt)
hist(lrt_p_values, breaks = 10)
```

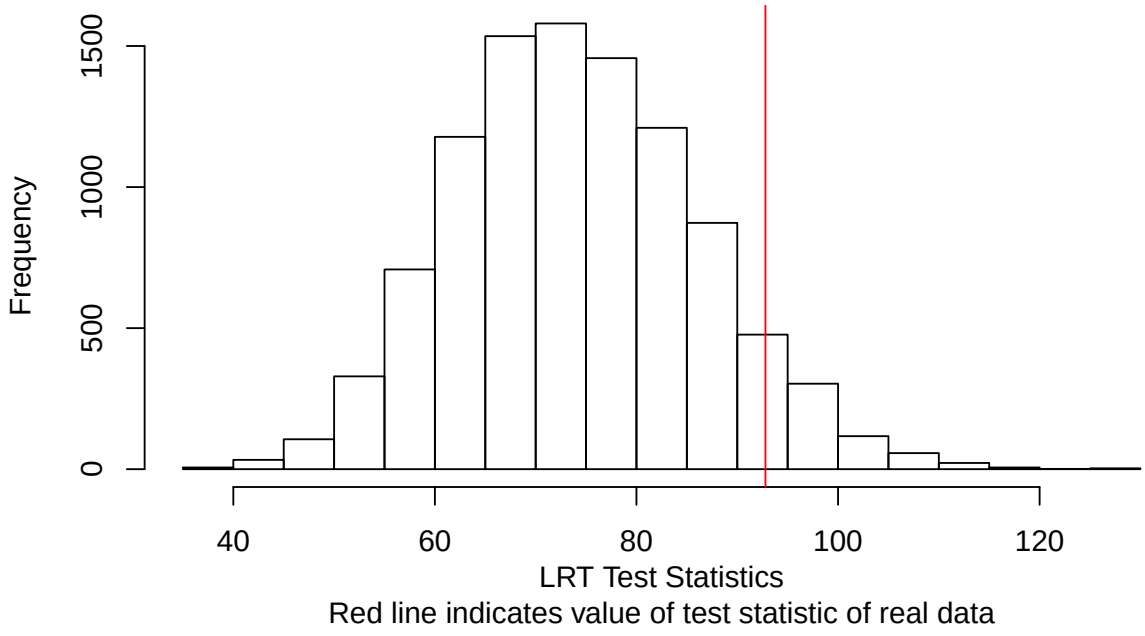


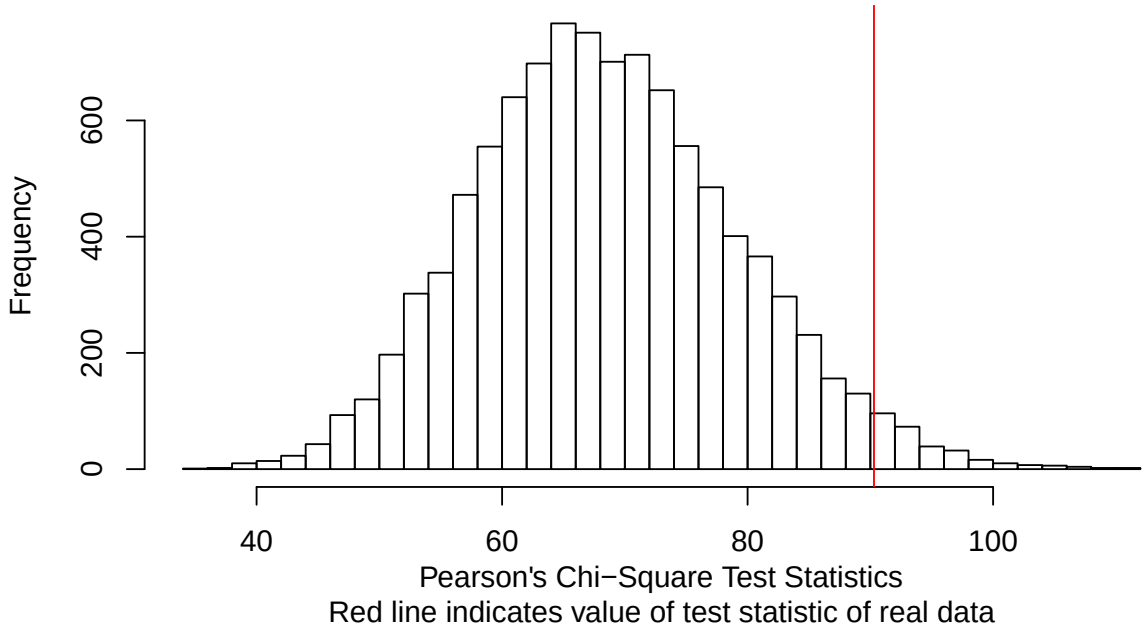
This is quick and dirty analysis, but it is clear from looking at the histogram of p-values that very low ones are generally uncommon. Given a p-value cutoff of .1, we would expect to see $36 * .1 = 3.6$ p-values below .1. In the above histogram, you see about 3 results with p-values below .1. This gives us no reason to doubt that simple binary Markov chains can be used to model the month of January in each year, as the years with low p-values may very well be the result of having data from a great many years.

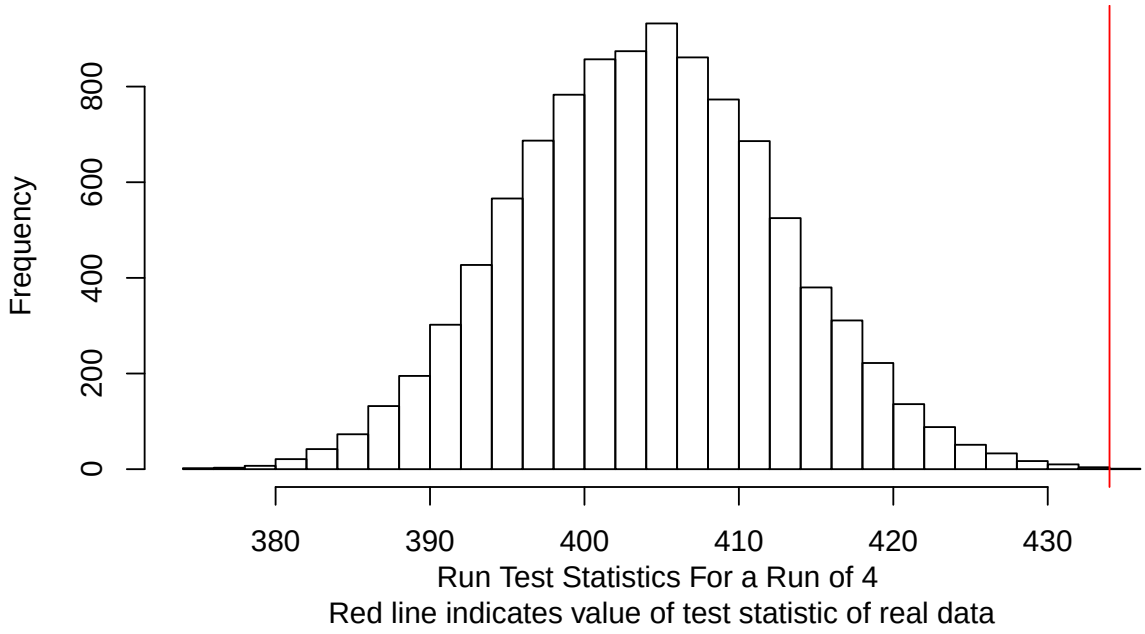
We're finally ready to call `multiple_binary_test()`, which will produce an object of class `multiple_binary_test`. Afterwards, we'll look at some plots and a summary of our data. Specifically, we'll be taking a look at the distribution of test statistics and the p-value for the likelihood ratio test (LRT). This test will give us a answer to the question of whether or not the null hypothesis of a single binary chain is acceptable against the alternative of a second order binary chain.

```
snoqualmie_test <- multiple_binary_test(binary_chains = snoqualmie_matrix, swaps = 10000,
                                        n = 10000)
plot(snoqualmie_test)
```

Visual representation of the actual Markov Chain.


Frequency
0
500
1000
1500
40
60
80
100
120
LRT Test Statistics
Red line indicates value of test statistic of real data


Frequency
0
200
400
600
40
60
80
100
Pearson's Chi−Square Test Statistics
Red line indicates value of test statistic of real data



By looking at these plots, we notice a few things. First, plotting a object of class `multiple_binary_test` gives us a whole lot of information, much of which isn't particularly interesting to us. We'll just be looking at the tile plot and the histogram of likelihood ratio test statistics. The tile plot lets us get a general feel for the makeup of our set of chains, and the histogram of likelihood ratio statistics shows us that the test statistic of the real set of chains lies somewhere in the meaty part of the distribution, making it unlikely that it is an unusual observation. For more detailed and precise analysis, let's turn our attention to the summary of `snoqualmie_test`.

```
summary(snoqualmie_test)
```

```
##
## Call:
## multiple_binary_test(binary_chains = snoqualmie_matrix, swaps = 10000,
##     n = 10000)
##
## Test statistics
##                Min       1Q        Med        3Q       Max      stat    Pr(>test stat)
## LRT       36.39     65.52     73.47     82.07    128.05     92.80    0.0682  .
## Chi Sq        34.00     61.05     67.93     75.47    111.32      90.3     0.027  *
## Run =  4       374       399       405       410       435       434     1e-04  ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## ---
```

Here you can see the structure of the `summary()` output. The likelihood ratio test statistics are broken down by quantile, and there is a p-value printed next to them as well. In this case, the p-value of the likelihood ratio test indicates that a single binary chain is probably acceptable for modeling this data.

References

Guttorp, P., and V.N. Minin. 1995. *Stochastic Modeling of Scientific Data*. Chapman & Hall/CRC Texts in Statistical Science. Taylor & Francis. https://books.google.com/books?id=LHdkc6tmF2EC.

2.3 Oklahoma Tornadoes

See next page...

Oklahoma Tornadoes

Conrad Cartmell

2016-05-31

In "A Markov Chain Model of Tornadic Activity" (Caren et al.), the authors propose that tornadoes in the United States can be modeled with Markov chains. In the paper, they encode their data in a matrix with each row representing a year from 1953 to 1998, and each column representing a day of the year. A 1 in an entry in the matrix indicates that at least one tornado occurred on the corresponding year and date and 0 in an entry indicates that no tornadoes occurred on the corresponding day.

If the presence of tornadoes in the greater United States can be modeled with a binary Markov chain, it stands to reason that tornadoes in Oklahoma could be modeled in the same fashion. The data encoded in the included data set `ok_tornado` is encoded in the same way as the data in the Drton et al. paper, but it covers only tornadoes in Oklahoma, and instead of including only the years 1953 to 1998, it includes all years between 1950 and 2015. It's time to get started.

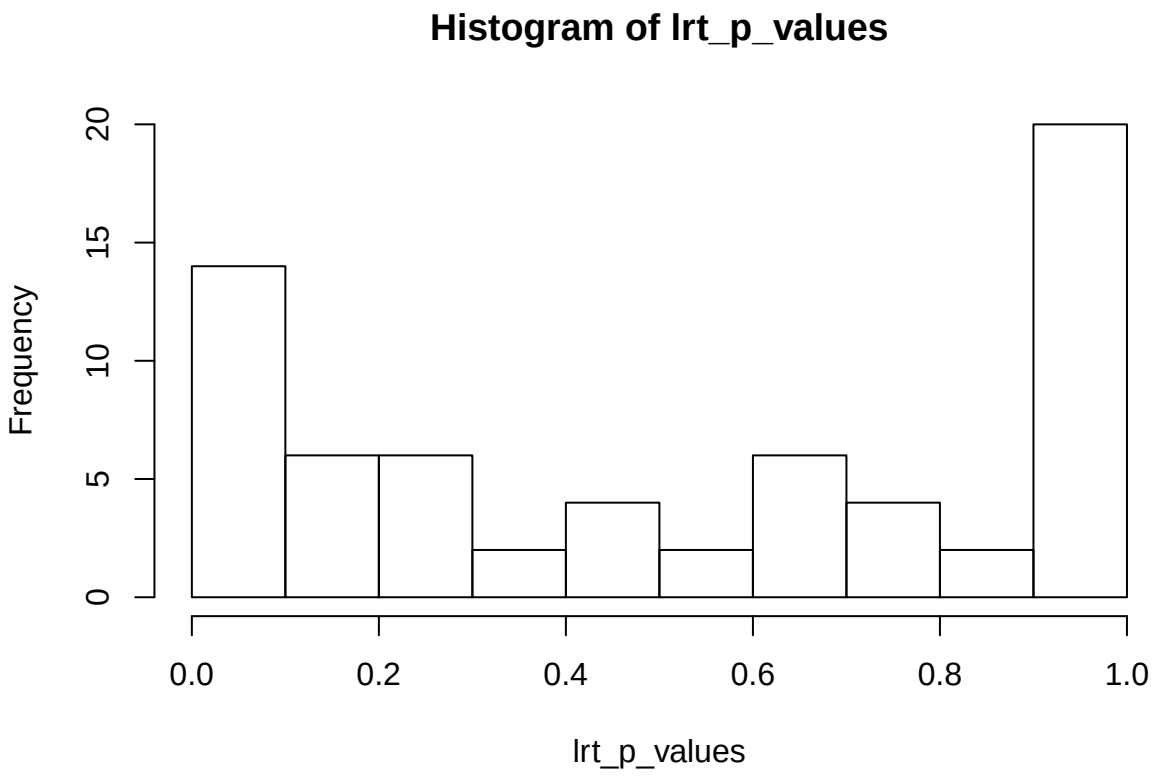
The first thing that we are going to investigate is whether every year is adequately modeled by a first order Markov chain. We'll run `single_binary_test()` on every single year that we have data, and compile a vector of likelihood ratio test statistic p-values to see if there are issues with using binary Markov chains for each one.

We need to take our data and transform it into forms that `single_binary_test()` and `multiple_binary_test()` can understand:

```
library(maRkov)
tornadoes <- as.matrix(ok_tornado)
```

Now that we have a vector representing the days with tornadoes in 2009 and a matrix representing days with tornadoes from 1950-2015, we'll run `single_binary_test()` on the chain from 2009 and look at a summary of the results:

```
lrt_p_values <- apply(tornadoes, 1,
                      function(X)(single_binary_test(binary_chain = X,
                                                     swaps = 10000, n = 10000))$p_value_lrt)
hist(lrt_p_values)
```

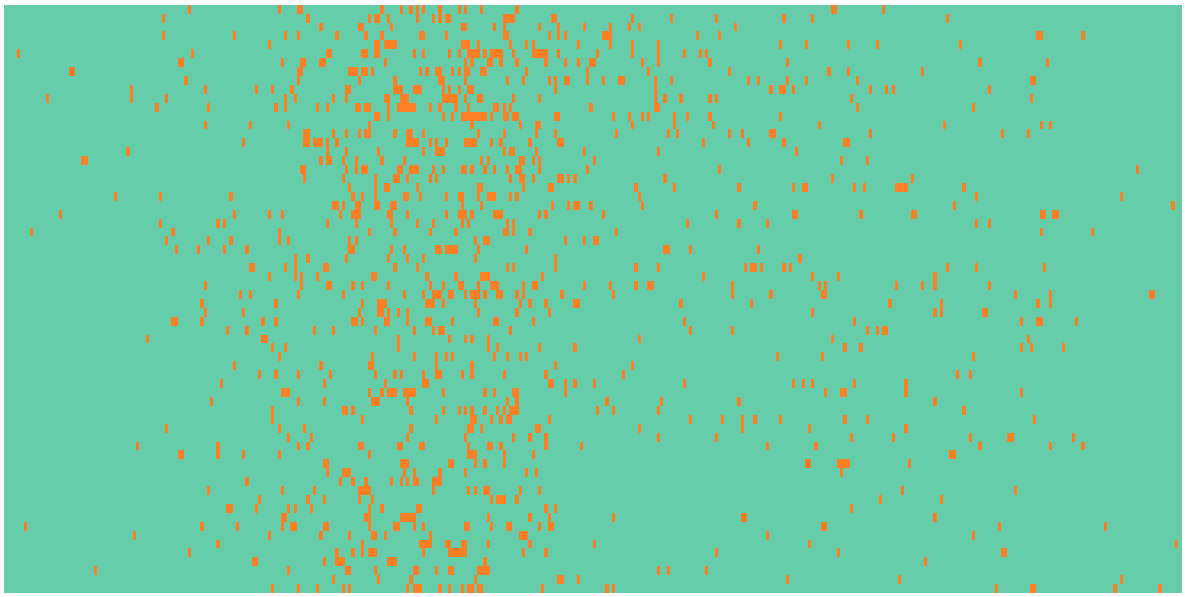


This is not good. As you can see from the histogram of p-values, we have ~14 p-values that indicate significance at the .1 level. We would expect to see $66 * .1 = 6.6$ if all the sequences could be adequately modeled by first order Markov chains. None the less, we'll forge on our investigation to see if we can identify what is going on.

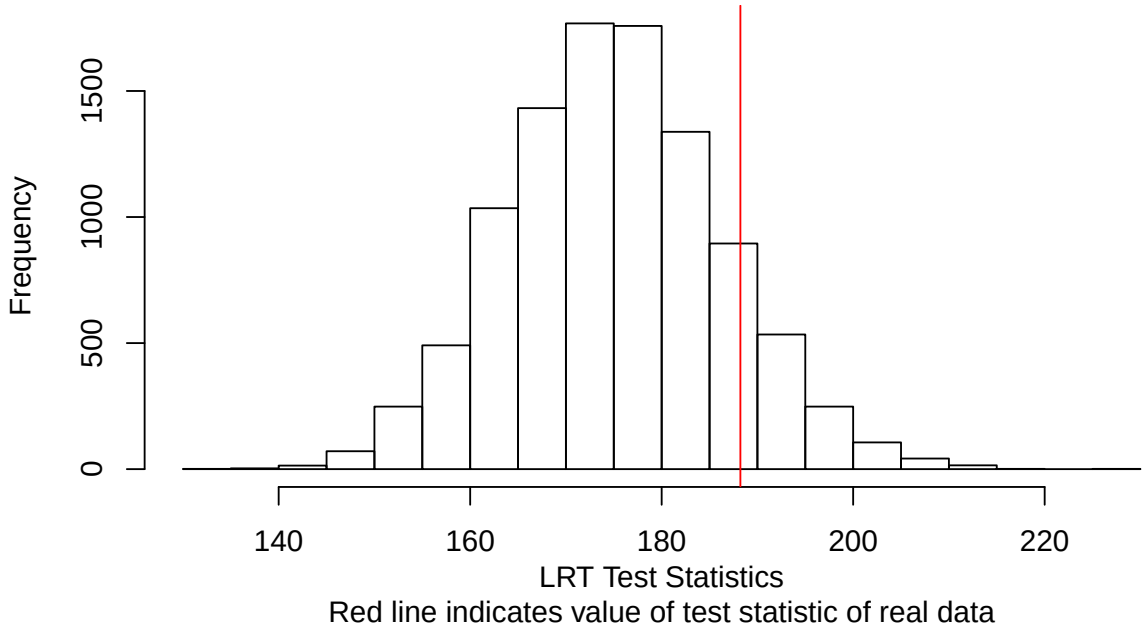
```
tornadoes_test <- multiple_binary_test(tornadoes, swaps = 10000, n = 10000)
```

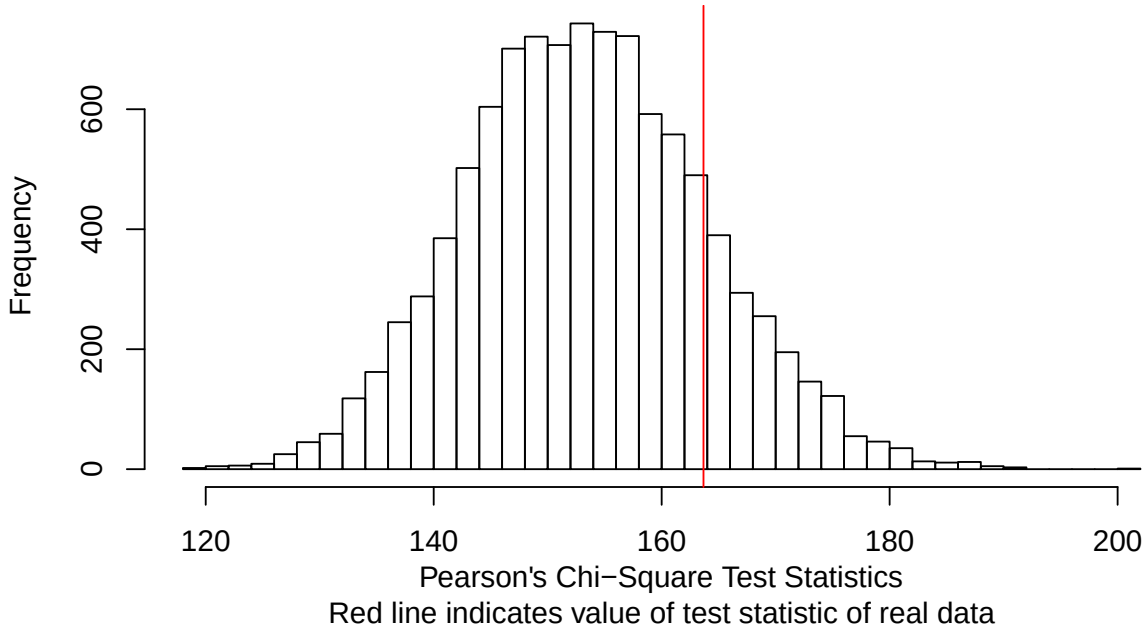
And then we'll plot the results:

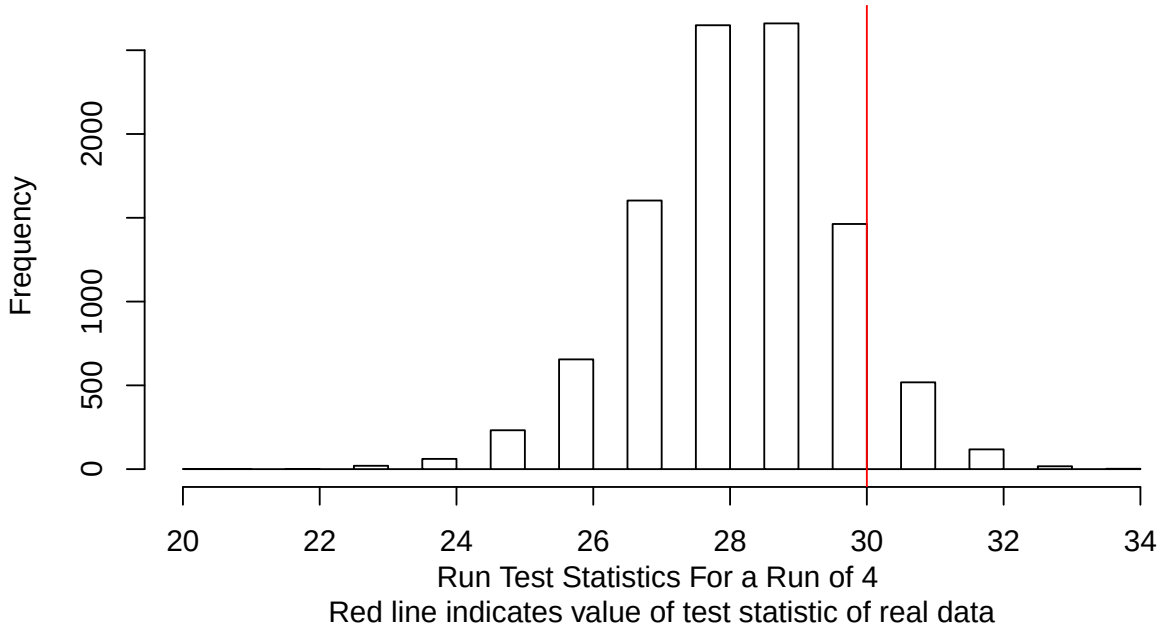
```
plot(tornadoes_test)
```



Visual representation of the actual Markov Chain.







As you can see from the likelihood ratio test statistic histogram, our data seems to be extremely unusual. Let's take a look at the summary of `tornadoes.test`:

```
summary(tornadoes_test)
```

```
##
## Call:
## multiple_binary_test(binary_chains = tornadoes, swaps = 10000,
##     n = 10000)
##
## Test statistics
##                Min       1Q        Med        3Q       Max      stat    Pr(>test stat)
## LRT       134.0     167.4     174.8     182.4     226.3     188.2     0.124
## Chi Sq        119.8     146.1     153.1     160.4     201.9     163.7     0.165
## Run =  4        20        27        28        29        34        30     0.212
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## ---
```

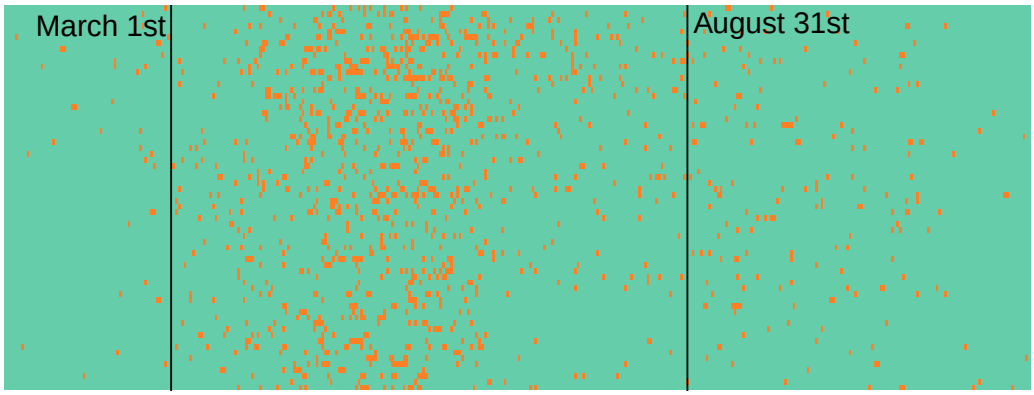
From this summary, we can see that our p-value for the likelihood ratio test statistics is quite small, so we can conclude that a simple first order Markov chain is probably not appropriate to model this data. Why could this be? if you look at the first plot generated by `plot(tornadoes_test)`, you will notice that there is a remarkable clustering effect in roughly the second quarter of the year. If you go to the website of Oklahoma City, and navigate to their tornado safety page, found at [http://www.okc.gov/safety/tornado/], you will find the following:

> When is tornado season?
>
> Tornado season is generally March through August, although tornadoes can occur at any time of the year.

Which suggests that days with tornadoes are going to be found predominantly between March 1st and August 31st, or between roughly the 60th and 244th days of the year. Let's see what our tile plot of Oklahoma tornadoes looks like when we superimpose those two dates upon it.

```
torn_data <- tornadoes_test$data[[1]]
reverse <- nrow(torn_data):1
torn_data <- torn_data[reverse, ]

image(c(1:ncol(torn_data)), c(1:nrow(torn_data)), t(torn_data),
      axes = FALSE,
      ylab = " ",
      xlab = "Visual representation of the actual Markov Chain.",
      col = c("aquamarine3", "chocolate1"))
segments(60, 0, 60, 67)
segments(244, 0, 244, 67)
text(35, 63, "March 1st")
text(274, 63, "August 31st")
```



Visual representation of the actual Markov Chain.

As you can see, there does seem to be marked difference between the distribution of tornadoes between these two dates and the distribution of tornadoes in the rest of the year. This fact begs the question: is it possible that we could model tornado season in Oklahoma with a Markov chain? Let's find out. We'll need to truncate our data to include only the days between March and August.

```
tornadoes_season <- tornadoes[ , 60 : 244]
tornadoes_season_test <- multiple_binary_test(tornadoes_season, swaps = 10000, n = 10000)
```

Now we'll look at the summary of the data, since we've already produced the tile plot.

```
summary(tornadoes_season_test)
```

```
##
## Call:
## multiple_binary_test(binary_chains = tornadoes_season, swaps = 10000,
##     n = 10000)
##
## Test statistics
##                Min       1Q        Med        3Q       Max      stat    Pr(>test stat)
## LRT       115.4     152.7     161.5     170.3     216.9     176.6     0.129
## Chi Sq        101.7     131.6     139.8     148.2     197.1     152.6     0.158
## Run =  4        12        23        25        26        34        28     0.117
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## ---
```

Again, it turns out that a simple, first order binary Markov chain does is not sufficient for modeling this data. Why is this?

It wasn't mentioned at the beginning of this vignette for the sake of constructing an interesting problem, but the authors of the Drton et al. paper did not actually use a first order binary Markov chain to describe

their data. In their case, they used what are called non-homogeneous Markov chains to describe their data. These are Markov chains for which the transition probabilities change as the chain moves through states. Given a little bit of thought, the requirement for this kind of chain makes perfect sense, as there are climate factors that fluctuate throughout the year that affect tornado formation. Another issue is that tornadoes in an area limited to the size of Oklahoma are affected by far more local weather patterns than tornadoes across the entire United States, which may make them harder to model with Markov chains. Unfortunately, this is the limit of the functionality of the package, so all we can conclude is that simple first order binary Markov chains are not sufficient to represent this data.

References

Caren, Mathias Drton, Mathias Drton, Peter Guttorp, Caren Marzban, and Joseph T. Schaefer. "A Markov Chain Model of Tornadic Activity." *Mon. Wea. Rev* 131: 2941–53.

2.4 Madras Schizophrenia Study

See next page...

Madras Schizophrenia Study

Conrad Cartmell

2016-05-29

The Madras Schizophrenia Study was a long term longitudinal study on the presence of schizophrenia in men and women in the Indian city of Madras, which is now known as Chennai. The data used in this vignette has been sourced from a book on longitudinal studies called "Analysis of Longitudinal Data" (Diggle, Liang, and Zeger 1994) that can be found at [http://faculty.washington.edu/heagerty/Books/AnalysisLongitudinal/].

The data found in `madras` is encoded as a data frame with 69 rows and 12 columns, with each row representing a different patient, and each column representing a month since initial hospitalization. A 1 represents a month in which symptoms were present, and a 0 represents one in which none were. The original dataset from the book featured 86 different individuals, both men and women, but not all of them were followed up on for 12 whole months. Since the program `multiple_binary_test()` requires rectangular data, these individuals were removed from the dataset, leaving us with the 69 individuals mentioned.

The first thing that we need to do is to transform our data into something that the maRkov package can understand. `multiple_binary_test()` takes a binary matrix as input, so that's what we will transform our data into:

```
library(maRkov)
madras <- as.matrix(madras)
head(madras)
```

```
##   Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
## 1   1   1   1   1   1   0   0   0   0   0   0   0
## 2   1   1   1   0   0   0   0   0   0   0   0   0
## 3   1   0   0   0   0   0   0   0   0   0   0   0
## 4   0   0   0   0   0   0   0   0   0   0   0   0
## 5   0   0   0   0   0   0   0   0   0   0   0   0
## 6   0   0   0   0   0   0   0   0   0   0   0   0
```

```
str(madras)
```

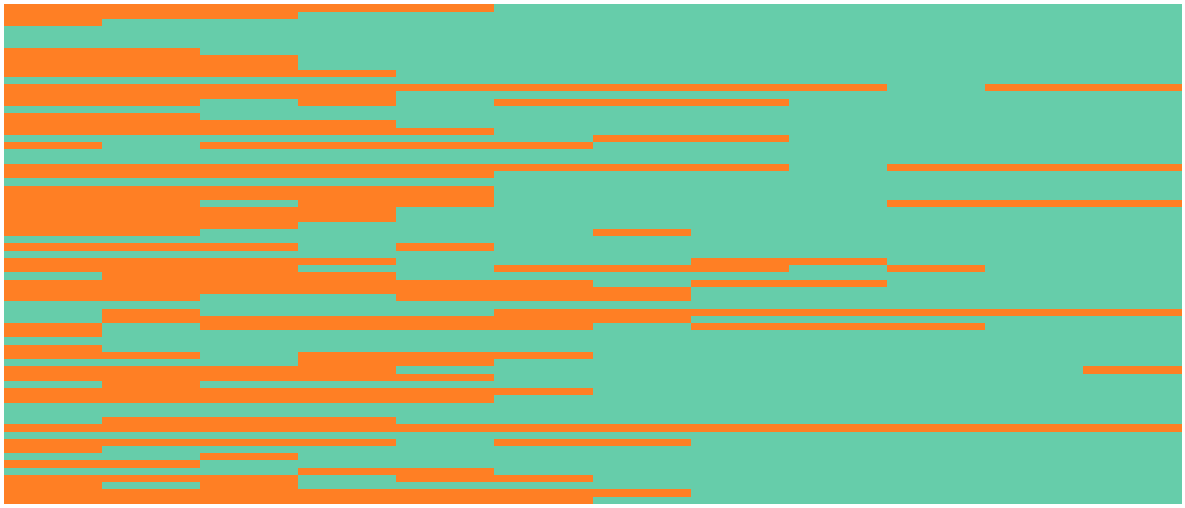
```
##  num [1:69, 1:12] 1 1 1 0 0 0 1 1 1 1 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:69] "1" "2" "3" "4" ...
##   ..$ : chr [1:12] "Jan" "Feb" "Mar" "Apr" ...
```

Looks great! Now, ideally we would want to examine each of these chains separately to see if a single binary Markov chains are appropriate models for each of them. Unfortunately, given the fact that chain is extremely short. However, since we have many of them, we can asses them all taken together to see if a single binary Markov model is appropriate when compared to multiple binary Markov chains.

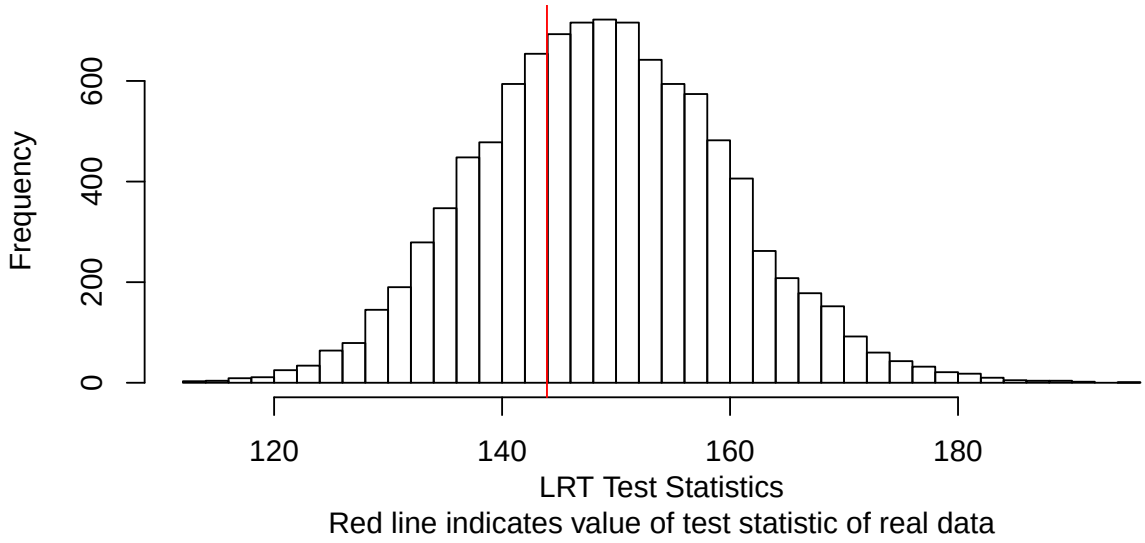
```
madras_test <- multiple_binary_test(madras, swaps = 10000, n = 10000)
```

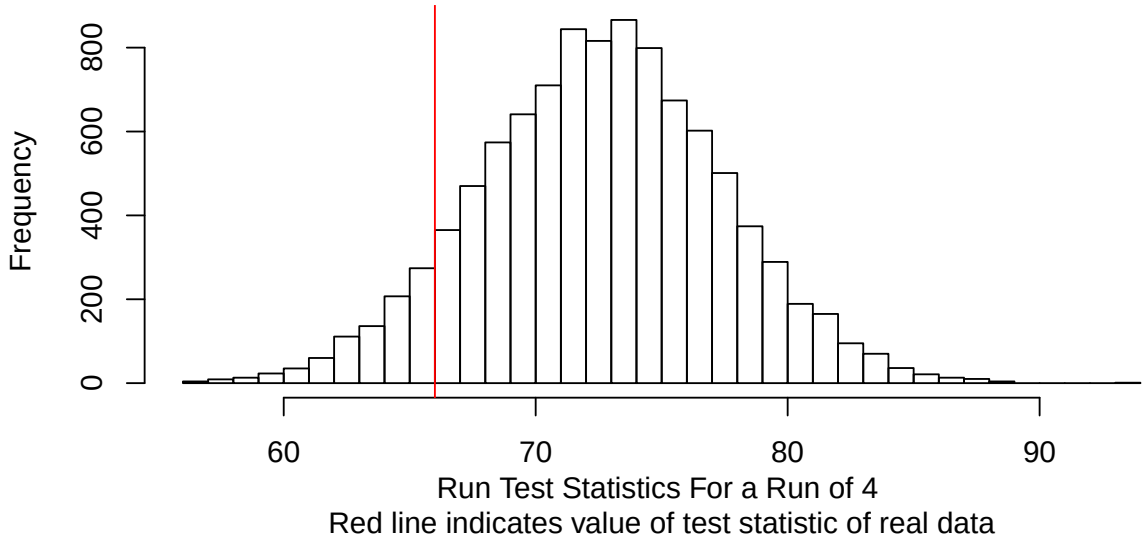
And then we'll plot our results to see if we can identify any patterns:

```
plot(madras_test)
```



Visual representation of the actual Markov Chain.





As you can see from the histogram plots, it does not seem that there is any significant issue with using one Markov chain to model all of the data. Let's look at a more detailed summary to be sure:

```
summary(madras_test)
```

```
##
## Call:
## multiple_binary_test(binary_chains = madras, swaps = 10000, n = 10000)
```

```
##
## Test statistics
##                 Min       1Q        Med        3Q       Max      stat    Pr(>test stat)
## LRT       112.2     141.4     148.7     156.2     194.4     143.9     0.665
## Chi Sq        74.43    107.76    118.87    131.33    193.94     151.6    0.0477  *
## Run =  4        56        70        73        76        94        66      0.94
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## ---
```

As you can see, the high likelihood ratio p-value suggests no reason to reject the hypothesis that a single Markov chain can be used instead of multiples.

The issue that exists with this analysis is that we removed the data with missing entries. In "Analysis of Longitudinal Data", the authors discuss the differences between the patients who dropped out and those who did not, and, as it turns out, the ones who drop out usually do so because their symptoms are not at all improving. In essence, this means that we are losing the representative nature of our sample by removing missing entries. Essentially then, the result that we get in this case means that, generally, people who have schizophrenia and are getting better tend to get better in similar ways.

References

Diggle, P., K.Y. Liang, and S.L. Zeger. 1994. *Analysis of Longitudinal Data.* Oxford Statistical Science Series. Clarendon Press. https://books.google.com/books?id=955qAAAAMAAJ.

3 Object Documentation

See next page...

Package 'maRkov'

May 27, 2016

Type Package

Title Goodness-of-fit Tests for Binary Markov Chains

Version 0.1

Date 2015-08-04

Author c(person(given = ``Conrad'', family = ``Cartmell'', role = c(``aut'', ``cre''), email = ``cwcartmell@gmail.com''), person(given = ``Debashis'', family = ``Mondal'', role = ``ctb'', email = ``debashis.stat@gmail.com''))

Maintainer Conrad Cartmell <cwcartmell@gmail.com>

Description Preforming goodness-of-fit tests on single binary Markov chains, as well as sets of multiple binary Markov chains. Provides LRT, chi square, and run test statistics, and generates data that can be used to preform user-created tests as well.

License GPL

Imports Rcpp

LinkingTo Rcpp

LazyData TRUE

RoxygenNote 5.0.1

Depends R (>= 2.10)

Suggests knitr, rmarkdown

VignetteBuilder knitr

R topics documented:

`alter_to_true_binary` *Take a one dimensional vector with two unique values and change those values to integers 0 and 1.*

Description

`alter_to_true_binary` takes a one dimensional vector that contains two unique values and switches all of the values with either integer values 0 or 1, so that the resulting vector has a one-to-one correspondence with the original.

Usage

```
alter_to_true_binary(bin_chain, success)
```

Arguments

bin_chain	A one dimensional vector with two unique elements.
success	Denotes the data entry to be counted for run statistics.

Details

For this function to work properly, its argument should be checked first using check_false_binary, and only used in alter_to_true_binary if check_false_binary returns TRUE.

Examples

```
alter_to_true_binary(c("A","B","B","B","A","B","A","A"))
alter_to_true_binary(c(TRUE,TRUE,TRUE,FALSE,FALSE,TRUE,FALSE,FALSE,TRUE,TRUE,
FALSE))
```

---

alter_to_true_binary_multiple

Take a two dimensional matrix with two unique values and change those values to integers 0 and 1.

Description

alter_to_true_binary_multiple takes a two dimensional matrix that contains two unique values and switches all of the values with either integer values 0 or 1, so that the resulting matrix has a one-to-one correspondence with the original.

Usage

```
alter_to_true_binary_multiple(bin_chains, success)
```

Arguments

bin_chains	A two dimensional vector with two unique elements.
success	Denotes the data entry to be counted for run statistics.

Details

For this function to work properly, its argument should be checked first using check_false_binary_multiple and only used in alter_to_true_binary if check_false_binary_multiple returns TRUE.

Examples

```
alter_to_true_binary_multiple(matrix(data = c("A","B","A","B","A","B","B",
"A","A"), ncol = 3))
alter_to_true_binary_multiple(matrix(data = c(TRUE,TRUE,TRUE,FALSE,FALSE,
TRUE,FALSE,FALSE,TRUE,TRUE,FALSE), ncol = 3))
```

check_false_binary *Check if a vector has two unique elements.*

Description

check_false_binary returns TRUE if there are two unique elements in its argument, and returns FALSE if there are not two unique elements in its argument.

Usage

```
check_false_binary(bin_chain)
```

Arguments

bin_chain	A one dimensional vector.

Details

This function is not designed to be used outside of this package. It is sufficiently simple as to be practically redundant to the user.

Examples

```
check_false_binary(c(1,0,0,1,0,0,0,1))
check_false_binary(c("A","B","B","B","A","B","A","A"))
check_false_binary(c(TRUE,TRUE,TRUE,FALSE,FALSE,TRUE,FALSE,FALSE,TRUE,TRUE,
FALSE))
```

check_false_binary_multiple *Check if a two dimensional matrix has two unique elements.*

Description

check_false_binary_multiple returns TRUE is there are two unique elements in its argument, and returns FALSE if there are not two unique elements in its argument.

Usage

```
check_false_binary_multiple(bin_chains)
```

Arguments

bin_chains	A two dimensional matrix

Details

This function checks every row and column element to see if they all share the same two values.

Examples

```
check_false_binary_multiple(matrix(data = c(1,0,1,0,1,0,0,1,1), ncol = 3))
check_false_binary_multiple(matrix(data = c("A","B","A","B","A","B","B","A",
"A"), ncol = 3))
check_false_binary_multiple(matrix(data = c(TRUE,TRUE,TRUE,FALSE,FALSE,TRUE,
FALSE,FALSE,TRUE,TRUE,FALSE), ncol = 3))
```

---

check_true_binary *Check if a one dimensional vector has only integer elements 0 and 1.*

Description

check_true_binary returns TRUE if all of the elements in the argument are either integers of value 0 or 1. check_true_binary returns FALSE if not all of the elements in the argument are integers of value 0 or 1.

Usage

```
check_true_binary(bin_chain)
```

Arguments

bin_chain	A one dimensional vector.

Details

This function checks every element of its argument to see if they contain either the integer values 0 or 1.

Examples

```
check_true_binary(c(1,0,0,1,0,0,0,1))
check_true_binary(c("A","B","B","B","A","B","A","A"))
check_true_binary(c(TRUE,TRUE,TRUE,FALSE,FALSE,TRUE,FALSE,FALSE,TRUE,TRUE,
FALSE))
```

---

check_true_binary_multiple *Check if a two dimensional matrix has only integer elements 0 and 1.*

Description

check_true_binary_multiple returns TRUE if all of the elements in its argument are integers 0 or 1, and FALSE if not all of the values in its argument are integers 0 or 1.

Usage

```
check_true_binary_multiple(bin_chains)
```

Arguments

bin_chains	A two dimensional matrix.

Details

This function checks every row and column element of its argument to see if they contain either the integer values 0 or 1.

Examples

```
check_true_binary_multiple(matrix(data = c(1,0,1,0,1,0,0,1,1), ncol = 3))
check_true_binary_multiple(matrix(data = c("A","B","A","B","A","B","B","A",
"A"), ncol = 3))
check_true_binary_multiple(matrix(data = c(TRUE,TRUE,TRUE,FALSE,FALSE,TRUE,
FALSE,FALSE,TRUE,TRUE,FALSE), ncol = 3))
```

---

chi_sq_test_stat *Calculates the Pearson's chi square test statistic for a single binary chain .*

Description

chi_sq_test_stat takes a binary chain of data and calculates the Pearson's chi square test statistic associated with it.

Usage

```
chi_sq_test_stat(bin_chain, n_chain_uniques)
```

Arguments

bin_chain	A single binary chain of data in the form of an integer vector.
n_chain_uniques	A integer value representing the number of unique values in bin_chain.

---

chi_sq_test_stat_array *Calculate the chi square test statistics for many single binary chains.*

Description

chi_sq_test_stat_array takes a two dimensional matrix of many binary chains of data and returns a numeric vector filled with a chi square test statistic for each of them.

Usage

```
chi_sq_test_stat_array(bin_chains, n_chain_uniques)
```

Arguments

bin_chains	A integer matrix of binary chains of data, with each row being a different chain.
n_chain_uniques	A integer value representing the number of unique values in bin_chains.

---

ik_dim_sum *Sum all the values in the first and third dimensions of a three dimensional integer vector.*

Description

ik_dim_sum takes a three dimensional integer vector, fixes the value of the second dimension, and then sums the values of all the entries in the vector with that value for its second dimension.

Usage

```
ik_dim_sum(n, j)
```

Arguments

n	A three dimensional integer vector.
j	An integer that is a valid index of the second dimension of n.

---

indicate_run *Indicate whether or not there is a run at a point in a integer vector.*

Description

indicate_run takes an integer vector bin_chain, and two integers, p and i, and tells the user if a run of length p starting and index i in the form of a Boolean value.

Usage

```
indicate_run(bin_chain, p, i)
```

Arguments

bin_chain	A binary chain of data in the form of an integer vector.
p	A integer value representing the length of the run to test for.
i	A integer value representing the location in bin_chain to test for a run starting at.

`i_dim_sum` *Sum all the values in the first dimension of a three dimensional integer vector.*

Description

`i_dim_sum` takes a three dimensional integer vector, fixes the values of the second and third dimensions, and then sums the values of all the entries in the vector with those two values for its second and third dimensions.

Usage

```
i_dim_sum(n, j, k)
```

Arguments

`n`	A three dimensional integer vector.
`j`	An integer that is a valid index of the second dimension of `n`.
`k`	An integer that is a valid index of the third dimension of `n`.

`jk_dim_sum` *Sum all the values in the second and third dimensions of a three dimensional integer vector.*

Description

`jkDimSum` takes a three dimensional integer vector, fixes the value of the first dimension, and then sums the values of all the entries in the vector with that value for its first dimension.

Usage

```
jk_dim_sum(n, i)
```

Arguments

`n`	A three dimensional integer vector.
`i`	An integer that is a valid index of the first dimension of `n`.

`j_dim_sum` *Sum all the values in the second dimension of a three dimensional integer vector.*

Description

`j_dim_sum` takes a three dimensional integer vector, fixes the values of the first and third dimensions, and then sums the values of all the entries in the vector with those two values for its first and third dimensions.

Usage

```
j_dim_sum(n, i, k)
```

Arguments

`n`	A three dimensional integer vector.
`i`	An integer that is a valid index of the first dimension of `n`.
`k`	An integer that is a valid index of the third dimension of `n`.

`k_dim_sum` *Sum all the values in the third dimension of a three dimensional integer vector.*

Description

`k_dim_sum` takes a three dimensional integer vector, fixes the values of the first and second dimensions, and then sums the values of all the entries in the vector with those two values for its first and second dimensions.

Usage

```
k_dim_sum(n, i, j)
```

Arguments

`n`	A three dimensional integer vector.
`i`	An integer that is a valid index of the first dimension of `n`.
`j`	An integer that is a valid index of the second dimension of `n`.

madras *Data from the Madras schizophrenia study*

Description

A data set documenting the presence of thought disorders across 12 months since hospitalization. Each column represents a single patient, and each row represents a separate month. Months with thought disorders are encoded as 1 and those without are encoded as 0. Data is sourced from "Analysis of Longitudinal Data" by Peter J. Diggle, Patrick J. Heagerty, Kung-Yee Liang, and Scott L. Zeger. It was retrieved from http://faculty.washington.edu/heagerty/Books/AnalysisLongitudinal/madras.data . Individuals who did not have 12 months of post hospitalization data were removed to make the data rectangular.

Usage

```
madras
```

Format

A data frame with 69 rows and 12 columns.

metropolis *Generate independent data from a single binary chain.*

Description

`metropolis` takes a single binary chain of data in the form of an integer vector and generates `b` new independent chains of data, placing all of them in an integer matrix with the original data in the first row.

Usage

```
metropolis(bin_chain, m, b)
```

Arguments

`bin_chain`	A single binary chain of data represented by an integer vector.
`m`	An integer representing the number of swaps to be attempted.
`b`	An integer representing the number of new chains of data to be generated.

Details

`metropolis` works by taking the supplied `bin_chain`, and attempting `m` swaps on it, only preforming a swap of elements if doing so maintains the number of transitions between states in the resulting chain. `metropolis` then takes the resulting chain, and attempts `m` swaps on it again, then saving the resulting vector in a new row of an output matrix. `metropolis` does this `b` times, each time saving the resulting vector. Once all of the new data has been generated, `metropolis` returns the newly built integer matrix, of which the first row is the original chain of data `bin_chain`.

multiple_binary_test *Perform goodness-of-fit tests on multiple binary chains.*

Description

multiple_binary_test is used to preform goodness-of-fit tests on multiple binary chains of data of the same length to see if a Markov chain model is appropriate.

Usage

```
multiple_binary_test(binary_chains, swaps = 1000, n = 1000, run = 4,
  bins = 30)
```

Arguments

binary_chains	A two dimensional matrix, in which there are two unique values.
swaps	A positive nonzero integer value for the number of swaps to be attempted on the chain. Larger values will tend to yield "more independent" data. Generally, the number of swaps should be much larger then the number of elements in the matrix binary_chains.
n	A positive nonzero integer representing the number of new sets of chains to be generated.
run	The length of the run to test for if one is interested in run test statistics.
bins	The number of bins to be displayed in histograms of test statistics when one plots objects generated by multiple_binary_test.
success	Denotes the data entry to be counted for run statistics.

Details

multiple_binary_test works by taking the supplied binary_chains parameter, counting the transitions between different elements, and then generating n new sets of chains with the same number of transitions. It generates these new sets of chains by attempting to swap random elements of the chains swaps times, only doing so if the attempted swap preserves the number of transitions between the two unique elements of the chains. multiple_binary_test then saves the chain generated by this process, then preforms a number of swaps equivalent to the value of swaps on that chain again, then recording the result in a new entry in a list of data. multiple_binary_test does this n times to generate the n new sets of chains. These new sets of chains are effectively independent of the original one.

Once multiple_binary_test has generated new data, it preforms various tests of that data. included in the function are the likelihood ratio test, the Pearson's chi square test, and a run test for a run of length specified by the argument run.

Value

multiple_binary_test returns a list of class "multiple_binary_test" with the following elements:

data, a list of matrices, the first of which is binary_chains, and the rest of which are the generated data.

test_stats_lrt, vector of likelihood ratio test statistics for each element of list data.

test_stats_chi_sq, a vector of Pearson's chi square test statistics for each element of list data.

test_stats_run, a vector of run test statistics for a run of length run for each element of list data.

p_value_lrt, the p-value of binary_chain, calculated exactly from the distribution of test_stats_lrt.

p_value_chi_sq, the p-value of binary_chain, calculated exactly from the distribution of test_stats_chi_sq.

p_value_run, the p-value of binary_chain, calculated exactly from the distribution of test_stats_run.

call, the function call.

bins, the number of bins specified in the function call.

run, the length of run specified in the function call.

Examples

```
data <- as.matrix(maRkov::snoqualmie)
foo <- multiple_binary_test(binary_chains = data, swaps = 10000, n = 10000,
                            run = 3, bins = 50)
```

---

multiple_chi_sq_test_stat

Calculate the Pearson's chi square test statistic for a set of binary chains of data.

Description

multiple_chi_sq_test_stat takes a two dimensional integer vector bin_chains in which each row represents a single binary chain of data, and calculates a Pearson's chi square test statistic for the entire set.

Usage

```
multiple_chi_sq_test_stat(bin_chains, n_chain_uniques)
```

Arguments

bin_chains	A two dimensional integer vector where each row is a separate binary chain of data.
n_chain_uniques	An integer value representing the number of unique elements in the set of chains bin_chains.

`multiple_chi_sq_test_stat_array` *Calculate Pearson's chi square test statistics for many sets of binary chains of data.*

Description

`multiple_chi_sq_test_stat_array` takes a three dimensional vector containing multiple sets of binary chains of data, and returns a numeric vector with entries corresponding to the Pearson's chi square test statistics of each set of binary chains of data.

Usage

```
multiple_chi_sq_test_stat_array(bin_chains, n_chain_uniques)
```

Arguments

`bin_chains`	A three dimensional vector containing sets of chains of binary data.
`n_chain_uniques`	An integer value representing the number of unique elements in the set of chains `bin_chains`.

`multiple_indicate_run` *Indicate whether or not a run of a certain length exists starting at a certain point.*

Description

`multiple_indicate_run` takes a single binary chain `binChain`, a valid index of that chain `i`, and a length of run `p` and tests whether or not a run of that length starts at index `i`.

Usage

```
multiple_indicate_run(binChain, p, i)
```

Arguments

`binChain`	A one dimensional integer vector representing a binary chain of data.
`p`	An integer representing the length of run to test for.
`i`	An integer representing a valid index of `binChain`.

multiple_metropolis *Generate independent data from a set of binary chains.*

Description

multiple_metropolis takes a set of binary chains of data in the form of an integer matrix and returns a three dimensional integer vector with with the first entry of the first dimension filled with the original set of binary chains and the rest filled with independent chains generated by multiple_metropolis.

Usage

```
multiple_metropolis(bin_chains, m, b)
```

Arguments

bin_chains	An integer matrix whose rows represent separate binary chains of data.
m	An integer value representing the number of swaps to be attempted.
b	An integer value representing the number of new sets of data to be generated.

Details

multiple_metropolis works by taking a supplied set of binary chains bin_chains and attempting a number m swaps on entries of those chains, only swapping if doing so maintains the number of transitions between states that existed in the initial set of chains bin_chains. After it does this, it repeats the process on the newly generated set of binary chains of data b times, each time saving the new set of chains in a three dimensional vector of data. The first entry of the first dimension of this vector is used to store the original set of binary chains bin_chains.

multiple_run_test_stat

Calculate the run test statistic for a set of binary chains of data and a run of a certain length.

Description

multiple_run_test_stat takes a two dimensional integer vector bin_chains in which each row represents a single binary chain of data, and calculates a run test statistic for a run of length p for the entire set.

Usage

```
multiple_run_test_stat(bin_chains, p)
```

Arguments

bin_chains	A two dimensional integer vector where each row is a separate binary chain of data.
p	An integer value representing the length of run to test for.

multiple_run_test_stat_array

Calculate run test statistics for many sets of chains of binary data.

Description

`multiple_run_test_stat_array` takes a three dimensional integer vector containing multiple sets of binary chains of data, and returns a numeric vector with entries corresponding to the run test statistics for runs of length p for each set of binary chains of data.

Usage

```
multiple_run_test_stat_array(bin_chains, p)
```

Arguments

`bin_chains`	A three dimensional integer vector containing sets of chains of binary data.
`p`	An integer representing the length of run to test for.

n_counts

Second order transition counts for a single binary chain.

Description

`n_counts` counts the number of second order transitions in a binary chain of data, then returns a three dimensional vector whose indices represent the type of transition, and whose values represent the number of times that each transition occurs in the chain.

Usage

```
n_counts(bin_chain, n_chain_uniques)
```

Arguments

`bin_chain`	An integer vector representing a chain of data.
`n_chain_uniques`	The number or unique values in the chain `bin_chains`, represented as an integer value.

n_counts_multiple *Second order transition counts for multiple binary chains.*

Description

`n_counts_multiple` counts the number of second order transitions in a integer matrix whose rows represent individual binary chains. It returns a three dimensional vector whose indices represent the type of transition, and whose values represent the number of times that each transition occurs in the set of chains.

Usage

```
n_counts_multiple(bin_chains, n_chain_uniques)
```

Arguments

`bin_chains`	A two dimensional integer vector, each of whose rows represents a single binary chain of data.
`n_chain_uniques`	The number of unique values in the set of chains `bin_chains`, represented as an integer value.

ok_tornado *Tornado data for Oklahoma, 1950 - 2015*

Description

A data set documenting the presence of tornadic activity in Oklahoma. Each row represents a year, starting with 1950, and ending in 2015, and each column is a day of that year. Each day on which a tornado occurred is marked encoded as a 1 and each day without tornadic activity is encoded as a 0. The data is inspired by, but not drawn from, the paper "A Markov Chain Model of Tornadic Activity" by Mathias Drton, Caren Marzban, Peter Guttorp, and Joseph T. Schaefer. As mentioned in their paper, "day 366 of a non-leap year is coded as a non-tornadic day. This is not expected to adversely affect the results." Data is sourced from the National Weather Service Weather Forecast Office in Norman, Oklahoma, and can be found at http://www.srh.noaa.gov/oun/?n=tornadodata-ok-monthlyannual .

Usage

```
ok_tornado
```

Format

A data frame with 66 rows and 366 columns.

plot.multiple_binary_test

Produce plots from objects of class "multiple_binary_test".

Description

Produces some interesting plots of data and test statistics of objects of class `multiple_binary_test`.

Usage

```
## S3 method for class 'multiple_binary_test'
plot(x, ...)
```

Arguments

`x`	An object of `class` `multiple_binary_test`.
`...`	Further arguments passed to or from other methods.

plot.single_binary_test

Produce plots from objects of class "single_binary_test".

Description

Produces some interesting plots of data and test statistics of objects of class `single_binary_test`.

Usage

```
## S3 method for class 'single_binary_test'
plot(x, ...)
```

Arguments

`x`	An object of `class` `single_binary_test`.
`...`	Further arguments passed to or from other methods.

`print.multiple_binary_test` *Print instructions for examining objects of class "multiple_binary_test".*

Description

Instruct user to use `summary` and `print` to examine objects of class `multiple_binary_test`.

Usage

```
## S3 method for class 'multiple_binary_test'
print(x, ...)
```

Arguments

`x`	An object of `class` `multiple_binary_test`.
`...`	Further arguments passed to or from other methods.

`print.single_binary_test` *Print instructions for examining objects of class "single_binary_test".*

Description

Instruct user to use `summary` and `print` to examine objects of class `single_binary_test`.

Usage

```
## S3 method for class 'single_binary_test'
print(x, ...)
```

Arguments

`x`	An object of `class` `single_binary_test`.
`...`	Further arguments passed to or from other methods.

run_test_stat *Calculate the run test statistic for a single binary chain.*

Description

run_test_stat takes an integer vector bin_chain of a chain of binary data, and a integer p representing the length of run to test for. It returns the run test stat for that chain of data.

Usage

```
run_test_stat(bin_chain, p)
```

Arguments

bin_chain	A binary chain of data in the form of an integer vector.
p	An integer greater than one representing the length of run to test for.

run_test_stat_array *Calculate run test statistics for many binary chains.*

Description

run_test_stat_array takes an integer matrix with each row denoting a binary chain of data and returns an integer vector with run test statistics for runs of length p corresponding to each binary chain.

Usage

```
run_test_stat_array(bin_chains, p)
```

Arguments

bin_chains	A two dimensional integer matrix with each row denoting a individual binary chain of data.
p	An integer value representing the length of run to test for.

`single_binary_test` *Perform goodness-of-fit tests on a single binary chain.*

Description

`single_binary_test` is used to preform goodness-of-fit tests on single binary chains of data to see if a Markov chain model is appropriate.

Usage

```
single_binary_test(binary_chain, swaps = 1000, n = 1000, run = 4,
  tiles = 30, bins = 30, success = NULL)
```

Arguments

`binary_chain`	A one dimensional vector with two unique values.
`swaps`	A positive nonzero integer value for the number of swaps to be attempted on the chain. Larger numbers will tend to yield "more independent" data. Generally, the number of swaps should be far greater than the length of `binary_chain`.
`n`	A positive nonzero integer value representing the number of new chains to be generated.
`run`	The length of run to test for if one is interested in run test statistics.
`tiles`	The number of chains to be represented in the tile plot when one plots objects generated by `single_binary_test`
`bins`	The number of bins to be displayed in histograms of test statistics when one plots objects generated by `single_binary_test`.
`success`	Denotes the data entry to be counted for run statistics.

Details

`single_binary_test` works by taking the supplied `binary_chain` parameter, counting the transitions between different elements, and then generating `n` new chains with the same number of transitions. It generates these new chains by attempting to swap random elements of the chain `swaps` times, only doing so if the attempted swap preserves the number of transitions between the two unique elements of the chain. `single_binary_test` then saves the chain generated by this process, then preforms a number of swaps equivalent to the value of `swaps` on that chain again, then recording the result in a matrix of new data. `single_binary_test` does this `n` times to generate the `n` new chains. These new chains are effectively independent of the original one.

Once `single_binary_test` has generated new data, it preforms various tests on that data. Included in the function are the likelihood ratio test, the Pearson's chi square test, and a run test for a run of length specified by the argument `run`.

Value

`single_binary_test` returns a list of class `"single_binary_test"` with the following elements:

`data`, a matrix of data with `binary_chain` in the first row, and the generated `n` rows of data in the following columns.

`test_stats_lrt`, a vector of likelihood ratio test statistics for each row of data in `data`.

test_stats_chi_sq, a vector of Pearson's chi square test statistics for each row of data in data.

test_stats_run, a vector of run test statistics for a run of length run for each row of data in data.

p_value_lrt, the p-value of binary_chain calculated exactly from the distribution of test_stats_lrt

p_value_chi_sq, the p-value of binary_chain calculated exactly from the distribution of test_stats_chi_sq

p_value_run, the p-value of binary_chain calculated exactly from the distribution of test_stats_run

call, the function call.

run, the length of run specified in the function call.

tiles, the number of tiles specified in the function call.

bins, the number of bins specified in the function call.

Examples

```
data <- c(1,1,1,1,1,1,1,1,1,1,1,1,0,1,1,1,0,1,1,1,1,1,1,1,1,1,0,0,0)
foo <- single_binary_test(binary_chain = data, swaps = 10000, n = 10000,
run = 4, tiles = 32, bins = 100, success = 1)
```

---

snoqualmie *Rainfall data for the month of January in Snoqualmie Falls, 1948 - 1983.*

Description

A data set documenting the presence of precipitation in Snoqualmie Falls Washington on every day of every January from 1948 to 1983. 1 indicates a day with >=0.01 inches of precipitation, 0 indicates a day with <0.01 inches of precipitation. Data sourced from Peter Guttorp's Stochastic Modeling of Scientific Data, 1995.

Usage

```
snoqualmie
```

Format

A data frame with 36 rows and 31 columns.

---

summary.multiple_binary_test *Produce a summary of objects of class "multiple_binary_test".*

Description

Prints test statistics and p-values for objects of class multiple_binary_test.

Usage

```
## S3 method for class 'multiple_binary_test'
summary(object, ...)
```

Arguments

object	An object of class multiple_binary_test.
...	Further arguments passed to or from other methods.

---

summary.single_binary_test *Produce a summary of objects of class "single_binary_test".*

Description

Prints test statistics, p-values, and provides sample data for objects of class single_binary_test.

Usage

```
## S3 method for class 'single_binary_test'
summary(object, ...)
```

Arguments

object	An object of class single_binary_test.
...	Further arguments passed to or from other methods.

---

swap *Swap elements of single binary chains*

Description

swap is used to swap elements of a single binary chain if doing so maintains the same number of transitions between the two states of that chain.

Usage

```
swap(bin_chain, m)
```

Arguments

bin_chain	A binary one dimensional integer vector.
m	A integer value representing the number of swaps to attempt.

Details

swap takes a one dimensional vector of integers bin_chain and an integer m. It attempts to swap elements of bin_chain m times, each time only completing the swap if it does not affect the number of transitions between states in the sequence.

swap_mult *Swap elements of multiple binary chains*

Description

swap_mult is used to swap elements of multiple binary chains if doing so maintains the same number of transitions between the two states of those chains.

Usage

```
swap_mult(bin_chains, m)
```

Arguments

bin_chains	A two dimensional integer vector with binary values.
m	A positive nonzero integer value for the attempted number of swaps to attempt on bin_chains.

Details

swap_mult works by taking a two dimensional integer vector bin_chains and m, a number of times to attempt swaps. It generates random integers which are valid indices of the two dimensional vector bin_chains and tries to swap the elements of the vector at the indices that it generates, only doing so if this preserves the total number of transitions between states. After attempting m swaps, swap_mult returns the new, freshly swapped two dimensional vector of binary chains.

u1_test_stat *Calculates the likelihood ratio test statistic for a single binary chain.*

Description

u1_test_stat takes a binary chain of data and calculates the likelihood ratio test statistic associated with it.

Usage

```
u1_test_stat(bin_chain, n_chain_uniques)
```

Arguments

bin_chain	A binary chain of data in the form of a one dimensional integer vector.
n_chain_uniques	A integer value representing the number of unique values in bin_chain.

u1_test_stat_array *Calculate likelihood ratio test statistics for many binary chains.*

Description

u1_test_stat_array takes an integer matrix with each row denoting a binary chain of data and returns an integer vector with likelihood ratio test statistics corresponding to each binary chain.

Usage

```
u1_test_stat_array(bin_chains, n_chain_uniques)
```

Arguments

bin_chains	A two dimensional integer matrix with each row denoting a individual binary chain of data.
n_chain_uniques	An integer value representing the number of unique values in the binary chains found in bin_chains.

u6_test_stat *Calculate the likelihood ratio test statistic for a set of binary chains of data.*

Description

u6_test_stat takes a two dimensional integer vector bin_chains in which each row represents a single binary chain of data, and calculates a likelihood ratio test statistic for the entire set.

Usage

```
u6_test_stat(bin_chains, n_chain_uniques)
```

Arguments

bin_chains	A two dimensional integer vector where each row is a separate binary chain of data.
n_chain_uniques	An integer value representing the number of unique elements in the set of chains bin_chains.

u6_test_stat_array — *Calculate likelihood ratio test statistics for many sets of binary chains of data.*

Description

`u6_test_stat_array` takes a three dimensional vector containing multiple sets of binary chains of data, and returns a numeric vector with entries corresponding to the likelihood ratio test statistics of each set of binary chains of data.

Usage

```
u6_test_stat_array(bin_chains, n_chain_uniques)
```

Arguments

`bin_chains`	A three dimensional vector containing sets of chains of binary data.
`n_chain_uniques`	An integer value representing the number of unique elements in the set of chains `bin_chains`.

vec_greater_than — *Find the number of entries in a vector greater or equal to the value of the first entry.*

Description

`vec_greater_than` counts the number of entries in a numeric vector `testStats` whose values are greater than that of the value of the first element of the vector.

Usage

```
vec_greater_than(testStats)
```

Arguments

`testStats`	A one dimensional numeric vector.

Index

References

Besag, J. and D. Mondal (2013). "Exact Goodness-of-Fit Tests for Markov Chains". In: *Biometrics* 69.2, pp. 488–496. ISSN: 1541-0420. DOI: 10.1111/biom.12009. URL: http://dx.doi.org/10.1111/biom.12009.

Caren, Mathias Drton et al. "A Markov Chain Model of Tornadic Activity". In: *Mon. Wea. Rev* 131, pp. 2941–2953.

Diggle, P., K.Y. Liang, and S.L. Zeger (1994). *Analysis of longitudinal data*. Oxford statistical science series. Clarendon Press. ISBN: 9780198522843. URL: https://books.google.com/books?id=955qAAAAMAAJ.

Guttorp, P. and V.N. Minin (1995). *Stochastic Modeling of Scientific Data*. Chapman & Hall/CRC Texts in Statistical Science. Taylor & Francis. ISBN: 9780412992810. URL: https://books.google.com/books?id=LHdkc6tmF2EC.