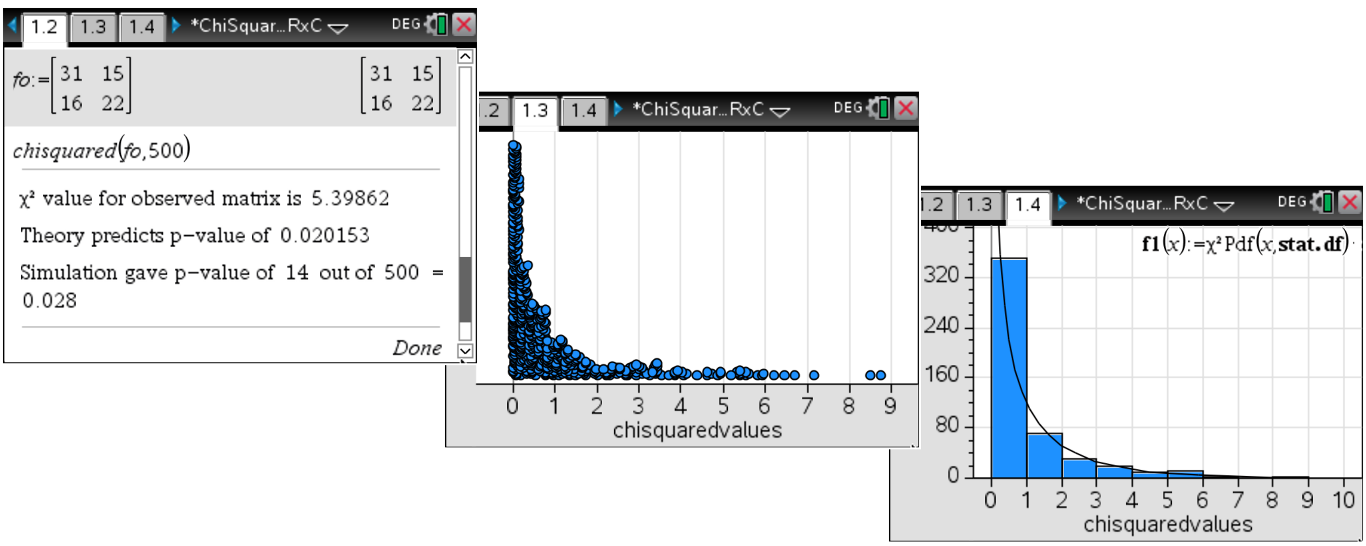
T³ Europe, Brussels – March 2017 – Nevil Hopley
Simulation of χ^2 Test of Association in a Two-Way Contingency Table

The following code simulates a test of association for any two-way contingency table and compares it to the appropriate χ^2 distribution.
It is made of two parts – a program [called ***chisquared***] which calls a function [called ***simulate***]
There are no restrictions on the size of the contingency table.
There are no checks on the magnitudes of the expected frequencies.
The following pages show the code, and then explain line-by-line how each part of the code works, with the help of an example.

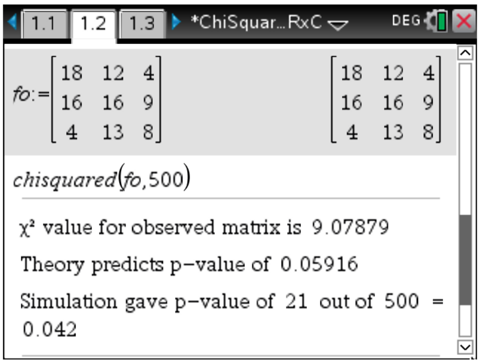
Line

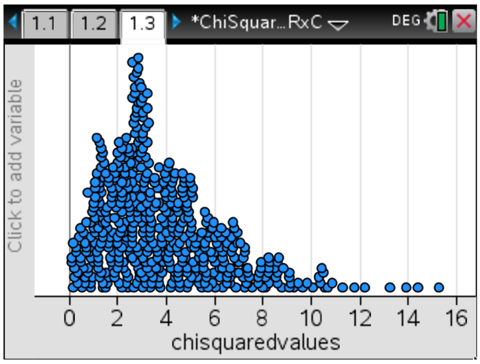
```
Define chisquared(matrix,trials)=
Prgm
Local coltotals,rowtotals,i,j,k,tally
coltotals:=mat▸list(sum(matrix))
rowtotals:=mat▸list(sum(matrixᵀ))
chisquaredvalues:={ }
For i,1,trials
tally:=0· matrix
For j,1,sum(coltotals)
tally:=tally+simulate(coltotals,rowtotals)
EndFor
χ²2way tally
chisquaredvalues:=augment(chisquaredvalues,{ stat.χ²})
EndFor
χ²2way matrix
k:=countIf(chisquaredvalues,?>stat.χ²)
Disp "χ² value for observed matrix is ",stat.χ²
Disp "Theory predicts p−value of ",stat.PVal
Disp "Simulation gave p−value of ",k," out of ",trials," = ",(k·1.)/trials
EndPrgm
```

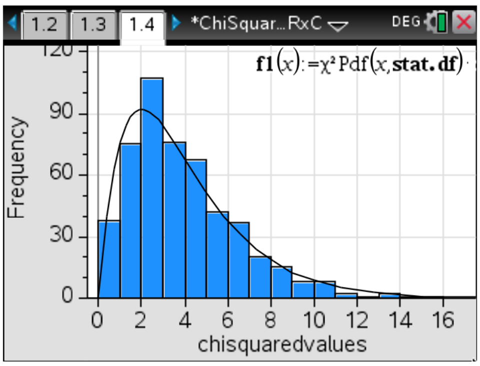


Line	Explanation	Example
1.	Defines input of observed frequency matrix and number of simulated trials	1.1 *Doc DEG *fo*:=$\begin{bmatrix}31&15\\16&22\end{bmatrix}$ $\begin{bmatrix}31&15\\16&22\end{bmatrix}$
2.	Start of Program	
3.	Define local variables so that they don't appear on the variable list after the program ends.	
4.	By default, the **sum( )** command returns a matrix of column totals. This is then converted to a list.	sum(*fo*) [47 37] *coltotals*:=mat▸list(sum(*fo*)) {47,37}
5.	Return a list of row totals by first taking the transpose of the observed frequency matrix.	sum(*fo*ᵀ) [46 38] *rowtotals*:=mat▸list(sum(*fo*ᵀ)) {46,38}
6.	Define the variable to hold the values of the chisquared statistic for each of the simulated trials.	*chisquaredvalues*:={ } { }
7.	Start of loop for the number of trials	
8.	Define a matrix that's the same dimensions as the observed frequency matrix, but contains all zeros.	*tally*:=0·*fo* $\begin{bmatrix}0&0\\0&0\end{bmatrix}$
9.	**sum(*coltotals*)** returns the total from whole matrix and this controls the number of times that ***simulate*** is called	sum(*coltotals*) 84
10.	The ***simulate*** function returns a matrix where all elements are zero and one element is 1. This is then added to the ***tally***	In effect, this line becomes…. *tally*:=*tally*+$\begin{bmatrix}0&1\\0&0\end{bmatrix}$ $\begin{bmatrix}0&1\\0&0\end{bmatrix}$
11.	End when the ***tally*** matrix has the same total as the observed frequency matrix.	For example, after 84 simulations…. *tally* $\begin{bmatrix}20&26\\27&11\end{bmatrix}$
12.	Conduct a chi-squared test on the ***tally*** matrix. The chi-squared statistic value is stored in the stat variable ***stat.***χ^2	χ²2way *tally*: *stat.results* "Title" "χ² 2−way Test" "χ²" 6.41996 "PVal" 0.011284 "df" 1. "ExpMatrix" "[...]" "CompMatrix" "[...]"
13.	Store the resulting chi-squared statistic for the simulated ***tally*** matrix	*chisquaredvalues* {6.41996}
14.	End after the specified number of trials, so that ***chisquaredvalues*** is then a long list of simulated values.	*chisquaredvalues* {6.41996,4.67321,2.78436,5.1184,3.11245▸
15.	Conduct a chi-squared test on the original observed frequency matrix, ***fo***	χ²2way *fo*: *stat.results* "Title" "χ² 2−way Test" "χ²" 5.39862 "PVal" 0.020153 "df" 1. "ExpMatrix" "[...]" "CompMatrix" "[...]"
16.	Count how many of the simulated chi-squared statistic values are greater than the statistic for the original observed frequency matrix	countIf(*chisquaredvalues*,?>5.39862) 2
17.	Display the chi-squared statistic for the observed frequency matrix	
18.	Display the theoretical p-value	
19.	Display the simulated result as a decimal, by "$k \times 1$."	
20.	End of program	

```
Define simulate(coltotals,rowtotals)=
Func
Local colbins,rowbins,colpath,rowpath,col,row
colbins:=augment({0},cumulativeSum(coltotals))
rowbins:=augment({0},cumulativeSum(rowtotals))
colpath:=sum(coltotals)· rand()
rowpath:=sum(rowtotals)· rand()
col:=0
While colpath>colbins[col+1]
col:=col+1
EndWhile
row:=0
While rowpath>rowbins[row+1]
row:=row+1
EndWhile
Return constructMat({1,row=r and col=c
                    {0, □                ,r,c,dim(rowtotals),dim(coltotals))
EndFunc
```







Line	Explanation	Example
1.	Defines inputs which are the lists: row-totals and column-totals. The size of the matrix is therefore given by the dimensions of these lists.	*rowtotals* {46,38} *coltotals* {47,37}
2.	Start of function	
3.	Define local variables so that they don't appear on the variable list after the function ends.	
4.	Define the column intervals into which simulated values will go. In this example, 0<value≤47 or 47<value≤84	cumulativeSum(*coltotals*) {47,84} augment({0},cumulativeSum(*coltotals*)) {0,47,84}
5.	And similarly for the rows. In this example, 0<value≤46 or 46<value≤84	augment({0},cumulativeSum(*rowtotals*)) {0,46,84}
6.	Define a random number between 0 and 84 that will then fall into either the first column or the second column. Here, as 0<12.3218≤47, it will go into column 1.	sum(*coltotals*) 84 sum(*coltotals*)· rand() 12.3218
7.	Define a random number between 0 and 84 that will then fall into either the first row or the second row. Here, as 46<61.6402≤84, it will go into row 2.	sum(*rowtotals*) 84 sum(*rowtotals*)· rand() 61.6402
8.	Set ***col*** variable for which column to put the value in.	
9.	Continue the loop when the value is greater than the next cutoff. Therefore, we exit the loop when it's not greater.	Compare 12.3218 to each of 47 and 84
10.	Increment column number by one	
11.	End loop	
12.	Set ***row*** variable for which row to put the value in	
13.	Continue the loop when the value is greater than the next cutoff. Therefore, we exit the loop when it's not greater.	Compare 61.6402 to each of 46 and 84
14.	Increment row number by one	
15.	End loop	
16.	Create a matrix with a 1 in the correct row and column, and 0's elsewhere. In this example the '1' is in ***row*** 2 and ***col*** 1.	$\text{constructMat}\left(\begin{cases}1, & 2=r \text{ and } 1=c \\ 0, & \text{Else}\end{cases}, r, c, 2, 2\right)$ $\begin{bmatrix}0 & 0 \\ 1 & 0\end{bmatrix}$
17.	End of function	

This function ensures that over sufficient repetitions, the correct proportions of 1's and 0's turn up in the matrix's elements, according to the row and column totals provided to it.

Authored by Nevil Hopley
February 2017
www.CalculatorSoftware.co.uk