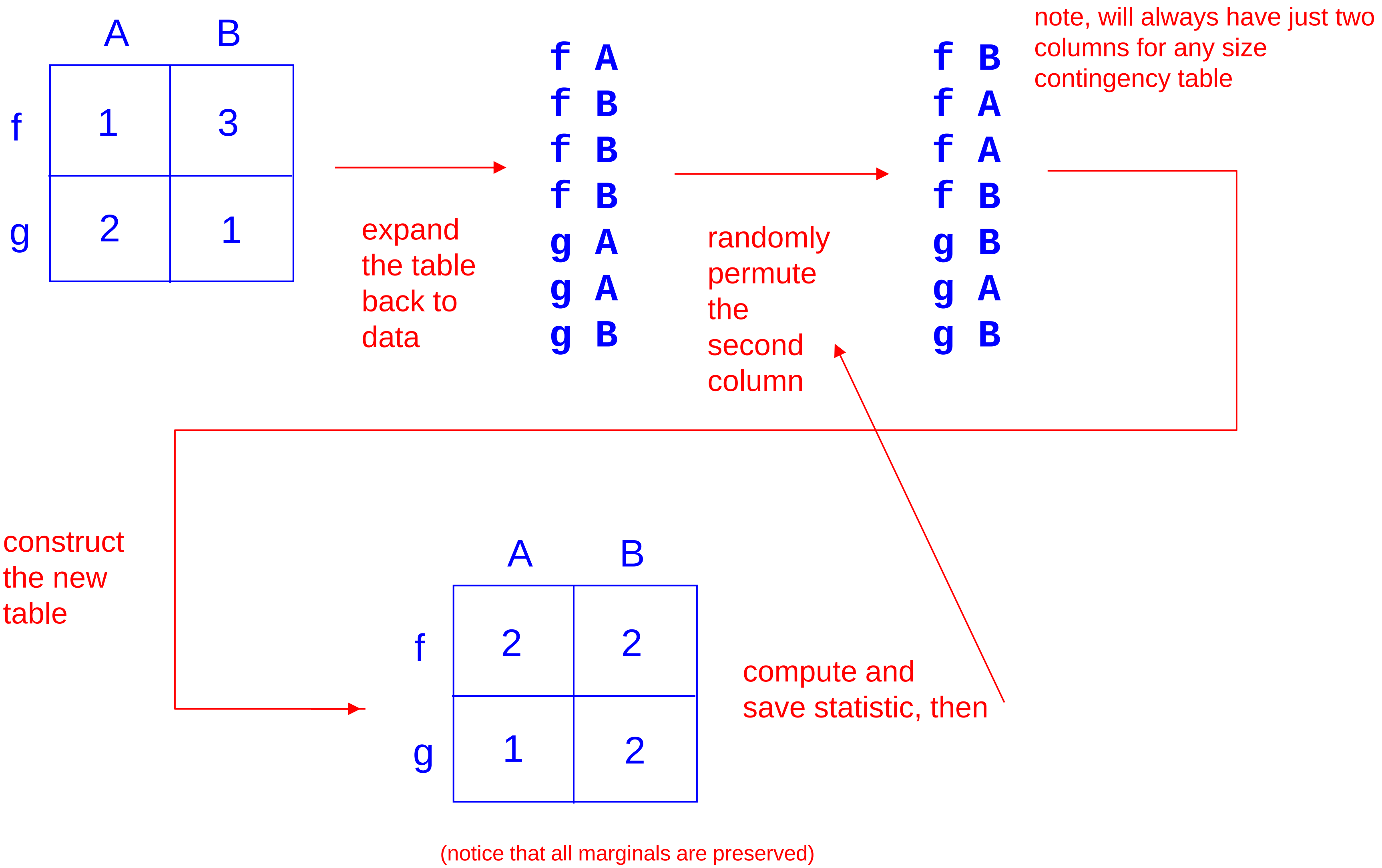
Opinionated Lessons in Statistics

by Bill Press

#34 Permutation Tests

An computational alternative to the Fisher Exact Test is the Permuation Test. The idea is to break any association between the row and column variables by shuffling. This is allowed under the null hypothesis of no association!



Aha! The permutation preserves all marginals. In fact, it is a Monte Carlo calculation of the Fisher Exact Test. And it is easy to compute for any size table!

	C_0	C_1
f_0	8	3
f_1	16	26

Code up the Wald statistic.

```
function t = wald(tab)
m = tab(1,1);
n = tab(1,2);
mm = m + tab(2,1);
nn = n + tab(2,2);
p1 = m/mm;
p2 = n/nn;
p = (m+n)/(mm+nn);
t = (p1-p2)/sqrt(p*(1-p)*(1/mm+1/nn));

table = [8 3; 16 26;]
table =
     8     3
    16    26

tdata = wald(table)
tdata =
       2.0542
```

The data show about a 2 standard deviation effect, except that they're not really standard deviations because of the small counts!

In scientific papers, people can equally well say, "Fisher Exact test" or "Permutation test". You might think that the former sounds more learned, but to me it sounds like they don't know exactly what their test actually did!

Expand the table and generate permutations:

```
[row col] = ndgrid(1:2,1:2);
d = [];
for k=1:numel(table); d = cat(1,d,repmat([row(k),col(k)],table(k),1)); end;
size(d)
ans =
    53     2
accumarray(d,1,[2,2])
ans =
     8     3
    16    26

gen = @(x) wald(accumarray( [d(randperm(size(d,1)),1) d(:,2)] ,1,[2,2]));

gen(1)
ans =
      -0.6676

perms = arrayfun(gen,1:100000);

hist(perms,(-4:.1:5))
cdfplot(perms)
```

This tells each cell its row and column number

(Darn it, I couldn't think of a way to do this in Matlab without an explicit loop, thus spoiling my no-loop record)*

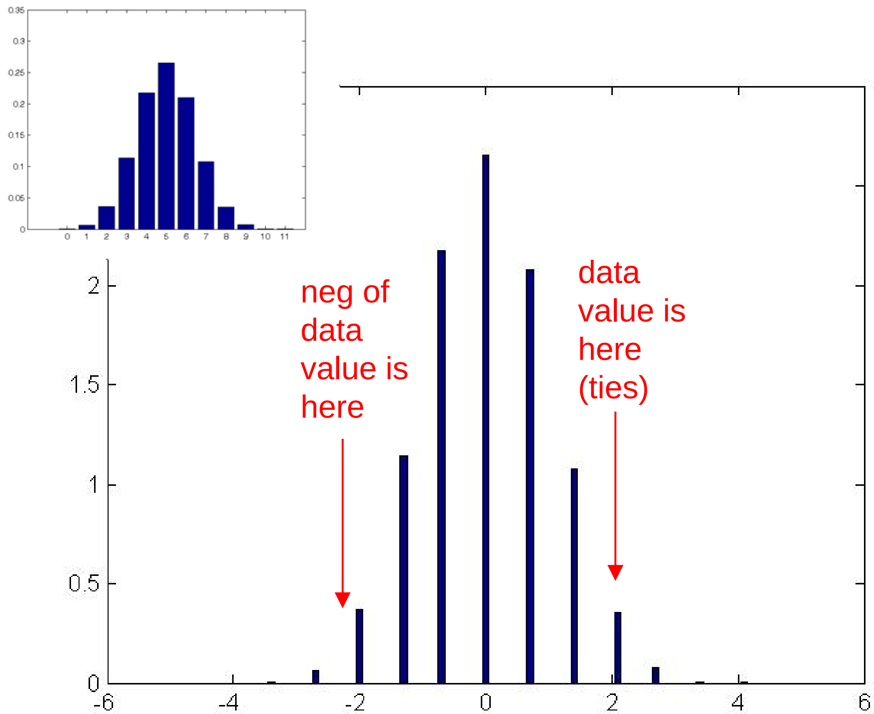
Check that we recover the original table.

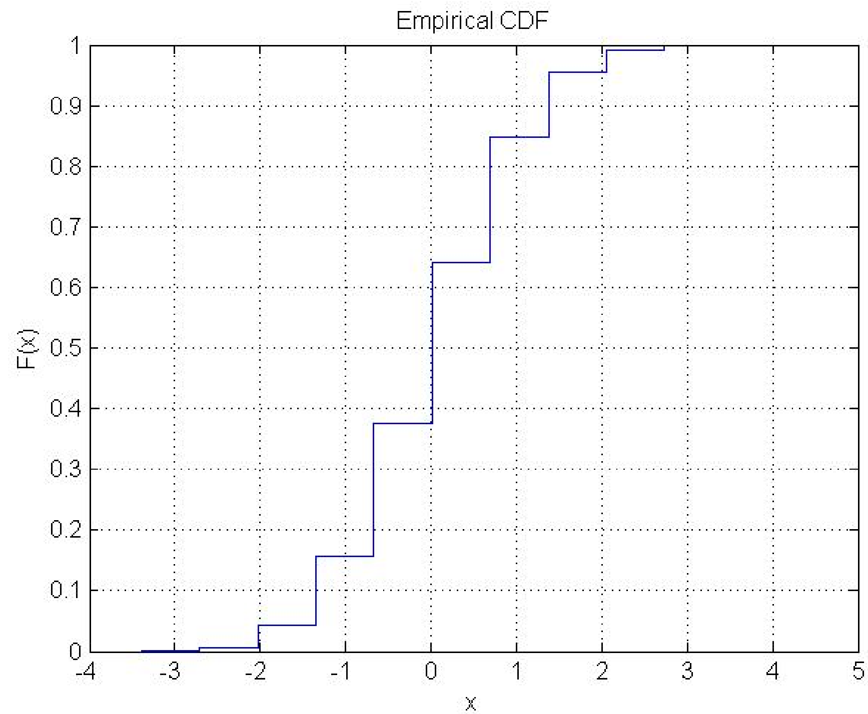
Try one permutation just to see it work.

It's fast, so can easily do lots of permuations.

*Peter Perkins (MathWorks) suggests the wonderfully obscure
`d = [rldecode(table,row) rldecode(table,col)];`
where rldecode is one of Peter Acklam's Matlab Tips and Tricks.

We get discrete values because only a few discrete tables are possible.






```
pvalgt = numel(perms(perms>wald(table)))/numel(perms)
pvalge = numel(perms(perms>=wald(table)))/numel(perms)
pvaltt = (numel(perms(perms > wald(table)))+numel(perms(perms < -wald(table))))/numel(perms)
pvaltte = (numel(perms(perms >= wald(table)))+numel(perms(perms <= -wald(table))))/numel(perms)
pvalwrongtail = numel(perms(perms> -wald(table)))/numel(perms)
```

```
pvalgt = 0.00812
pvalge = 0.04382
pvaltt = 0.01498
pvaltte = 0.05068
pvalwrongtail = 0.99314
```

We reproduce values from Fisher Exact test done combinatorially

To learn more, let's play with the first contingency table we looked at:

TABLE 1
Maternal drinking and congenital malformations

Malformation	Alcohol consumption (average no. of drinks/day) 0	< 1	1–2	3–5	≥ 6
Absent	17,066	14,464	788	126	37
Present	48	38	5	1	1

Source: Graubard and Korn (1987).

Different "standard methods" applied to this data get p-values ranging from 0.005 to 0.190. (Agresti 1992)

Fisher Exact Test done combinatorially is not a viable option (both because of computational workload and because we only derived the 2x2 case!)

So we'll try the (equivalent) permutation test.

Expand the table and generate 1000 permutations

(Now takes ~1 min. Go figure out how to do the permutation test without expanding all the data!)

```
table = [17066 14464 788 126 37; 48 38 5 1 1]
table =
       17066       14464         788         126          37
          48          38           5           1           1

pearson(table)
ans =
   12.0821

[row col] = ndgrid(1:size(table,1),1:size(table,2));
d = [];
for k=1:numel(table); d = cat(1,d,repmat([row(k),col(k)],table(k),1)); end;
size(d)
ans =
       32574
           2
tablecheck = accumarray(d,1,size(table))
tablecheck =
       17066       14464         788         126          37
          48          38           5           1           1

gen = @(x) pearson( accumarray([ d(randperm(size(d,1)),1) d(:,2)],1,size(table)));
gen(1)
ans =
    1.3378

perms = arrayfun(gen, 1:1000);
```

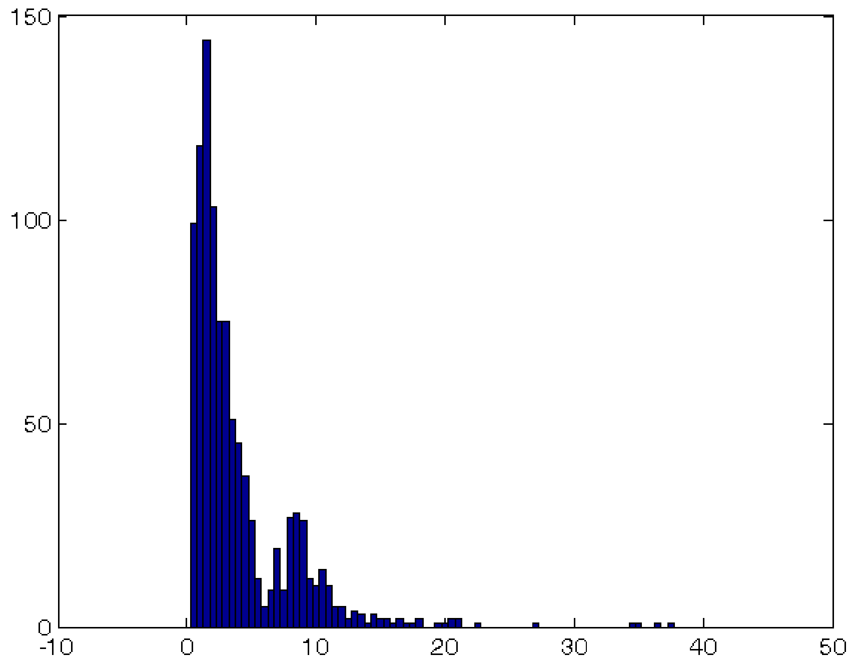
Yes, has the dimensions we expect.

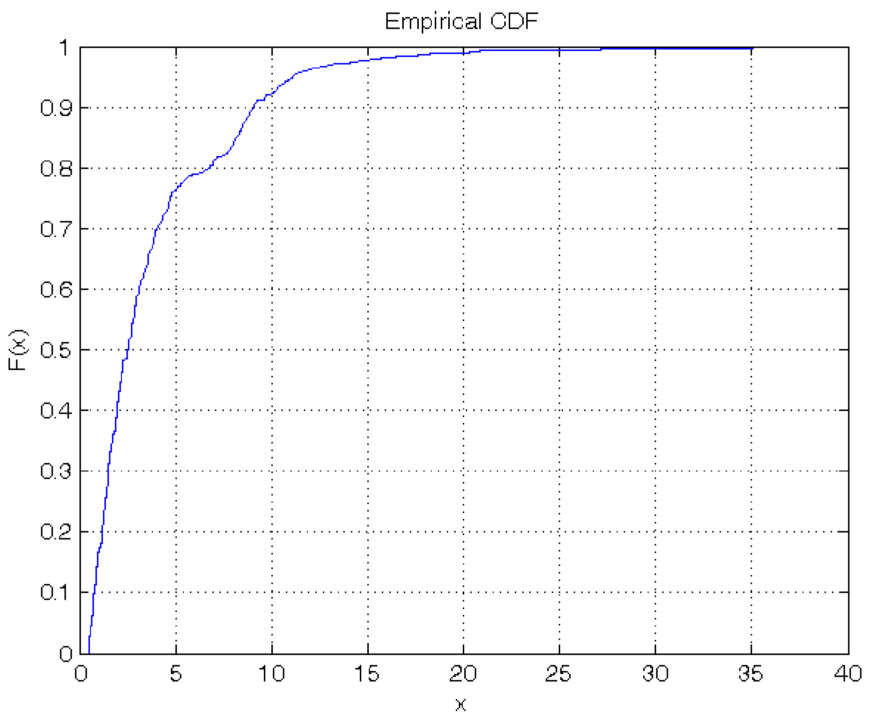
And we can reconstruct the original table.

```
hist(perms,(0:.5:40))

cdfplot(perms)

pvalgt = numel(perms(perms>pearson(table)))/numel(perms)
pvalge = numel(perms(perms>=pearson(table)))/numel(perms)
pvalgt =
    0.0380
pvalge =
    0.0380
```

our answer: the p-value

```
pearson(table)
ans =
   12.0821
```





Two questions remain:

1. How good or bad an approximation was it to hold all marginals fixed?
2. Is there a more powerful statistical test for this data?