

Counting data

Thus far, we have encountered data that were the results of measurements made on some process. Often, however, data take the form of counts of things, where a “thing” could be a quality, category, or any similar non-numeric division. For example, we might count the number of dendritic branchings that occurred on each of 5 types of growth media. The data (# of branchings) might look like the following:

Medium:	A	B	C	D	E
# of branchings:	12	15	20	19	26

Obviously, we probably have scientific reasons for believing that one or more growth media will do better than the others, but a statistical analysis allows us to compute the probability that the observed pattern of results could be due to chance alone. The total number of branchings is $N=92$ so, intuitively, we would expect about $92/5$ branchings in each group if chance alone were operating. Of course, we won't always get $92/5$ branchings per group because of the very chance fluctuations with which we are concerned (in fact, we will never get $92/5$ branchings because $92/5 = 18.4$ and counts, by definition, are integers).

In each group we observe O_g branches and, hence, $N - O_g$ “non-branches”. If chance alone is operating and all growth media are equally supportive, the number of counts O_g in each cell on a given experiment would be a random number from a binomial distribution with $N = 92$ and $p = 1/5$. To simulate how 92 branches would spread over 5 growth media by chance alone, we will generate 92 uniformly distributed random numbers and then count how many numbers are in each of 5 even intervals between 0 and 1:

```
branchloc = rand(92,1);
simcounts(1) = sum(branchloc < 1/5);
simcounts(2) = sum(branchloc > 1/5 & branchloc < 2/5);
simcounts(3) = sum(branchloc > 2/5 & branchloc < 3/5);
simcounts(4) = sum(branchloc > 3/5 & branchloc < 4/5);
simcounts(5) = sum(branchloc > 4/5);
```

Or alternatively:

```
branchloc = rand(92,1);
ngroup = 5;
for g=1:ngroup
    simcounts(g) = sum(branchloc > (g-1)/ngroup & branchloc < g/ngroup);
end
```

That's it! We now have simulated data from an experiment in which chance alone is distributing the number of branches among cells given 92 total branchings. To do a Monte Carlo simulation of this "world" in which chance alone is operating, we need merely repeat this several times over. But what we need now is some *test statistic* that we can compute on each simulated experiment (to develop a sampling distribution) and that we can also compute on our real data, and then see where the statistic for the data falls relative to the sampling distribution.

We know that the expected value in each cell is $E = N*1/5 = 18.4$ so a reasonable statistic could be formed by computing the difference between each cell count (real or simulated) and E , squaring it (so the negative differences don't cancel out the positive ones), and summing this up to compute a sum-squared-error (which by this time should be very familiar to us!).

An implementation in MATLAB is:

```
ngroup = 5;
totCount = 92;
E = totCount / ngroup;
nrep = 1000;
simcounts = zeros(ngroup,1);
simerrs = zeros(nrep,1);

for i=1:nrep
    branchloc = rand(totCount,1);
    for g=1:ngroup
        simcounts(g) = sum(branchloc > (g-1)/ngroup & branchloc < g/ngroup);
    end
    simerrs(i) = sum((simcounts - E).^2);
end

realcounts = [12, 15, 20, 19, 26];
realerr = sum((realcounts - E).^2);

hist(simerrs);
line([realerr realerr], [0 300]);
p = sum(simerrs >= realerr)/nrep;
disp(p);
```

The resulting plot and p-value that we got is below:

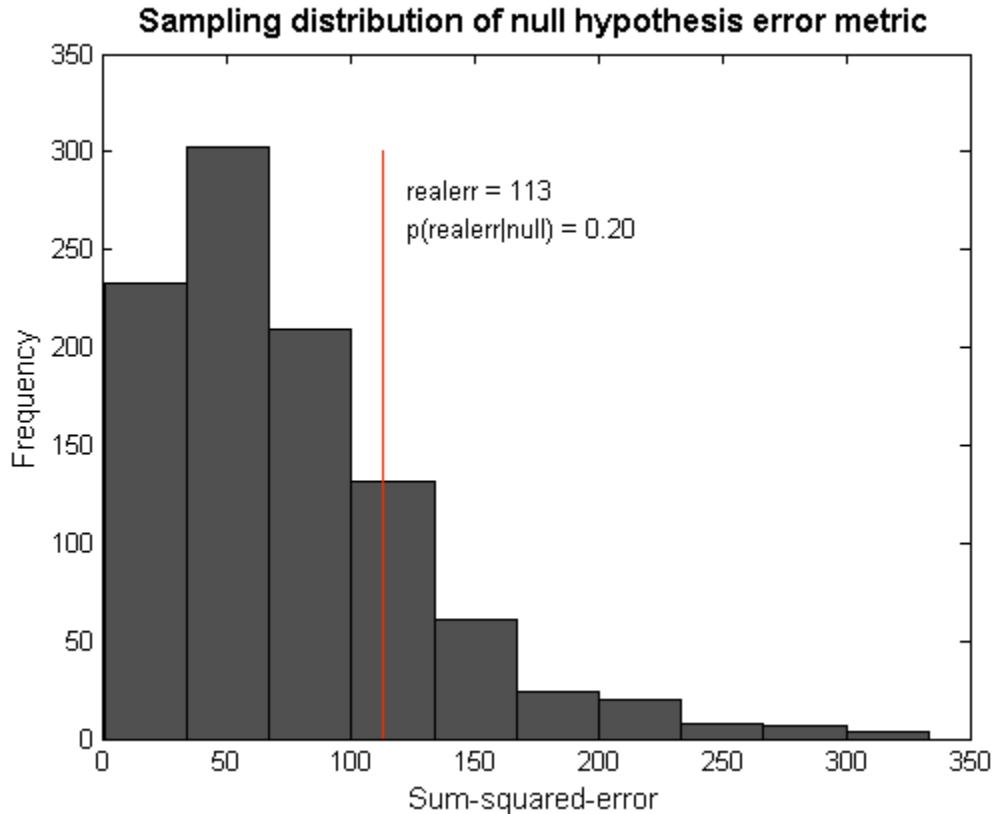


Figure 7.1. Sampling distribution of sum-square-error metric under the null assumption of equally supportive growth media. The observed error (113) is well within the distribution.

Let's return to our error metric, which is simply a sum-squared-difference. If we give it the arbitrary name ζ (zeta), it would be written as mathematically as:

$$\zeta = \sum_{i=1}^k (O_i - E_i)^2 = \sum_{i=1}^k (O_i - N/k)^2$$

O_i is the number of observed events (branchings, in this case) in each group, N is the total number of events, and k is the number of groups. Clearly, the distribution of this metric is going to change with both the number of groups, k (the more groups we sum over, the bigger we expect the metric to be overall), and the total number of things we are counting, N . Intuitively, if we are counting how many times a neuron spiked in one of two conditions over a 10 sec interval, and the total number of spikes was ~ 3000 , then a discrepancy of 10 spikes between observed and expected would be relatively small. If, on the other hand, we were counting the number of times a rat chose to press each of two

possible levers across 20 total trials, then a discrepancy of 10 between observed and expected would be the maximum discrepancy possible.

We can modify our metric above very easily to standardize across all possible values of N by dividing the squared difference by the expected value, and in so doing, we yield the traditional chi-squared “goodness-of-fit” test:

$$X^2 = \sum_{i=1}^k \frac{(O_i - E_i)^2}{E_i}$$

The reason that the expected value, and not its square, or the binomial variance, $N(1/k)(1 - 1/k)$, etc. is shown for the special case of $k=2$ in the appendix.

Another common use of the chi-squared test is to assess the independence of two variables. Let us introduce this use with a somewhat whimsical example. Suppose a street entertainer approaches you and offers you the following wager. He has two decks of 40 cards each. One deck consists of 20 Kings and 20 Jokers, while the other consists of 20 Queens and 20 Jokers. You will pay him a dollar to play, and he will deal out 40 2-card hands, one card from each deck. If there are more than 10 royal couples (a King and a Queen), he keeps the dollar. If, on the other hand, there are 10 or fewer royal couples, he gives you two dollars. Assuming the game is fair, we can easily figure out what we should expect to happen (on average). For each hand, there should be a 0.5 probability of getting a King from the first deck, and a 0.5 probability of getting a Queen from the second deck. Thus, if the two cards for a given hand and truly drawn independently, then the probability of getting the hand (K,Q) is $p(K)*p(Q) = 0.5*0.5 = 0.25$. Over 40 hands, then, the *expected value* for the number of occurrences of (K,Q) is $0.25*40 = 10$.

This situation is summarized below in a *contingency table*. The first deck is represented in the rows and the second deck is represented in the columns such that each possible hand (or “contingency”) corresponds to one of the central four cells. Entered in these cells are the expected values, or most likely outcomes, for each hand.

		Deck 2		Totals:
		Q	J	
Deck 1	K	.5*.5*40=(10)	(10)	20
	J	(10)	(10)	20
Totals:		20	20	Grand Tot. = 40

If the game is in fact fair, then it looks like a good deal for you, because you win on the most likely value and all those less than it, while your adversary only wins on values greater than the most likely one. Obviously, we expect some variability due to chance, but at some point, if the number of (K,Q) hands was too high, we would begin to suspect cheating. At what point would this be? How many is “too many”? Notice that in this

context – and this is an important point to understand – “cheating” corresponds to a *lack of independence* between the two decks, such that the presence of a King can be partially predicted by the presence of a Queen (and vice-versa).

Suppose the hands are dealt and there are a total of 14 royal couples. The *chi-squared contingency table* is then written as:

		Deck 2		Totals:
		Q	J	
Deck 1	K	14 (10)	6 (10)	20
	J	6 (10)	14 (10)	20
Totals:		20	20	Grand Tot. = 40

In this table, the observed values are in the central cells, along with the expected values (in parentheses). Notice that because there are a fixed number of each kind of card in each deck (given by the numbers in the margins of the table, or the *marginal totals*), once we know that there are 14 (K,Q) hands, the observed numbers of each of the other hands is known also. In other words, once one observed value is known, all the others are as well; in statistical terms, we have one *degree of freedom*.

We can now easily compute an error metric just as we did above for the simple (one variable) case. If we compute it as chi-squared (sum of $(O-E)^2$ over E), we get 6.4 as our answer. Now, as always in inferential statistical testing, the question becomes: what does this mean? What do we compare the value “6.4” to? The answer is, as always, that we need a sampling distribution for our error metric under the assumption that the two decks are independent of one another (i.e., that our entertainer is honest).

To simulate this sampling distribution, we only need to simulate an honest dealer. In other words, we can simply simulate the game many times, with a random shuffling of the two decks between each game. Let’s do this in MATLAB.

First, we’ll set up our two decks:

```
d1 = [ones(1,20), zeros(1,20)]; % use “1” for K and “0” for J
d2 = [ones(1,20), zeros(1,20)]; % use “1” for Q and “0” for J
```

Now we can easily shuffle them using the `sample()` function:

```
shuffled1 = randsample(d1, 40)
shuffled2 = randsample(d2, 40)
```

Notice that, unlike with our previous uses of `randsample()`, we have left the default value of ‘replace’ set to the default value (False). `randsample()` thus returns a complete copy of our decks, all 40 cards, but in a random order. Now we can “deal” and count the number of royal couples:

```
ncouples = sum(shuffled1 & shuffled2)
```

The “and” expression inside the brackets creates a vector whose value is “1” when both our shuffled decks contain a “1” (king from deck 1 and queen from deck 2) and zero everywhere else. Summing this vector thus gives the total number of royal couples. For the purposes of this game, we are only interested in the count (number of royal couples) in a single cell. All we have to do to obtain the “honest” sampling distribution is to embed and slightly modify the above statements so they run inside a loop and accumulate the count of interest. Here’s a short MATLAB program to accomplish this:

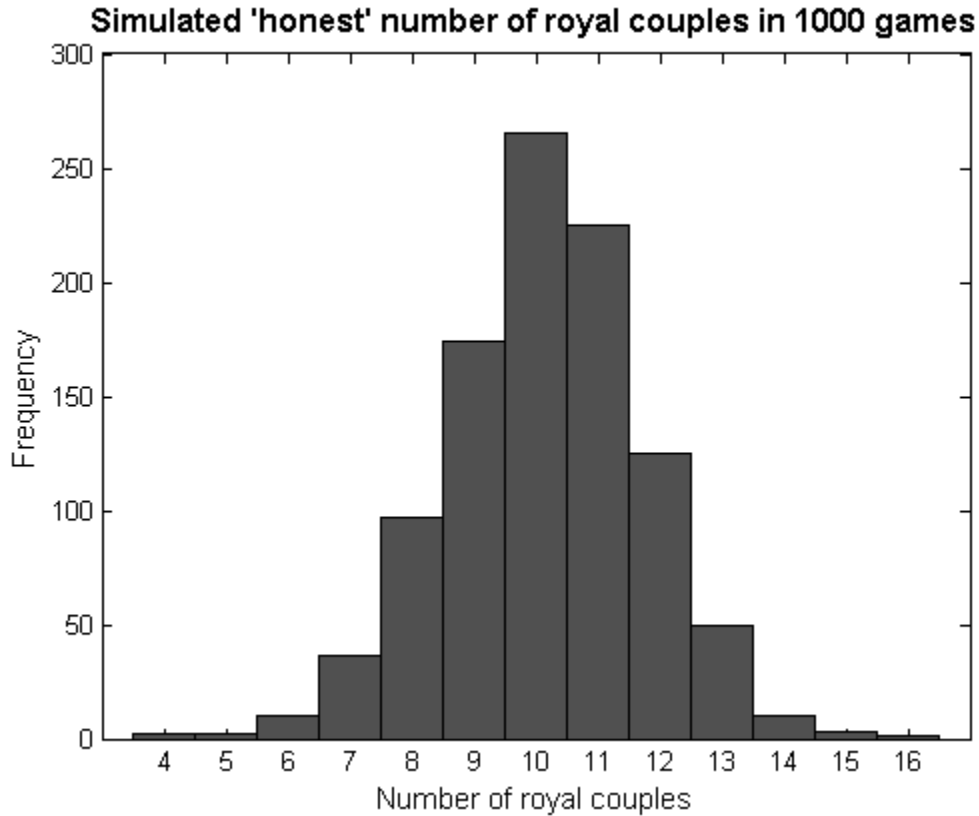
```
nrep = 1000;
nsimcouples = zeros(nrep,1);

d1 = [ones(1,20), zeros(1,20)]; % use “1” for K and “0” for J
d2 = [ones(1,20), zeros(1,20)]; % use “1” for Q and “0” for J

for i = 1:nrep
    shuffled1 = randsample(d1, 40);
    shuffled2 = randsample(d2, 40);
    nsimcouples(i) = sum(shuffled1 & shuffled2);
end

hist(nsimcouples)
```

The last line produces this output:



Hmm – it looks like 14 couples is a pretty rare event assuming that the game is fair! To figure out precisely how unlikely, all we have to do is count the number of times 14 or greater occurred and express it as a proportion of the total number of simulated games we played, which was 2000:

```
sum(nsimcouples >=14)/nrep
```

and this produces the a value of 0.017 – pretty rare indeed!

Now let us look at a slightly less trivial example by extending our original dendritic branching example. Below is a table of data, to which has been added two different growth factors (X and Y). A glance at the data reveals that growth factor Y is probably more effective at producing branching overall (how would you verify this?), but the question we wish to answer is “is the effect of growth factor independent of the medium?” A positive answer would indicate the presence of factor-medium combinations that are particularly good or bad at producing branching (over and above what would be expected by the simple additive effects of independent contributions).

Medium	
--------	--

		A	B	C	D	E	Marginal totals:
Growth Factor	X	12	15	20	19	26	92
	Y	25	21	41	31	32	150
	Marginal totals:	37	36	61	50	58	242

As above, the steps to answer this question are few and fairly straightforward:

- compute the counts we would expect assuming independence
- compute a sensible error metric between the observed and expected counts
- simulate a number of experiments assuming independence
- compare the step #2 result with the simulated distribution of the error metric

Luckily, MATLAB has a number of handy commands to make implementing these steps very easy. `realcounts = [12, 15, 20, 19, 26; 25, 21, 41, 31, 32];`

The result is a matrix corresponding the ten data values from the above summary table:

```
realcounts =
    12  15  20  19  26
    25  21  41  31  32
```

Next, we use the `sum()` function to compute the marginal totals, which we need to compute our expected values:

```
colsums = sum(realcounts);    % marginal sum for growth media
rowsums = sum(realcounts,2);  % marginal sum for growth factors
totalsum = sum(sum(realcounts));
```

The `sum()` function operates by default on columns (summing across rows), but can operate on rows (i.e. sum across the second dimension – columns) if supplied with the additional argument.

Now we can compute our table of expected values using the formula:

$$\text{“expected value} = (\text{row total} * \text{column total}) / \text{grand total”}$$

To understand how this formula computes the expected values, it may be useful to consider what the expected values in a 2 dimensional table actually mean. If we expect

independence of factors, we expect that the proportion of branchings grown in each medium is the same for both growth factors, or equivalently, that the proportion of branchings grown under each growth factor is the same regardless of the growth medium. For instance, consider expected value of branches grown in medium A with growth factor X. We expect the same proportion of the marginal total number of branchings in medium A (37) should be found in the dish with growth factor X as is the proportion of all branchings grown in any medium ($92/242$). The expected value for cell XA is then $37*(92/242)$, which is equivalent to $(37*92)/242$, the above formula.

To do this, we can use a nested loop to compute the expected values and enter them into a table:

```
E = zeros(2,5);    % expected values

for i=1:2
    for j=1:5
        E(i,j) = colsums(j).*rowsums(i)./totalsum;
    end
end
```

Alternatively, we use Matlab's strength in matrix algebra to avoid the nested for loops:

```
E = rowsums*colsums/totalsum;
```

Note the use of matrix multiplication “*” for the two marginal vector sums as opposed to scalar multiplication “.*” when operating cell-by-cell in the nested loops. If you are not sure about matrix multiplication, just stick with the nested loops method.

Half our steps are now done, so let us do our simulated experiments under the assumption that medium and growth factor are independent. First, we will set up our “decks” as in the card game example:

```
gf = [ repmat(1,rowsums(1),1); repmat(2,rowsums(2),1)];
gm = [ repmat(1,colsums(1),1); repmat(2,colsums(2),1); repmat(3,colsums(3),1);
      repmat(4,colsums(4),1); repmat(5,colsums(5),1)];
```

Now we can run our simulated experiments as we did in the card game example. To make our lives easier, however, we will use the `crosstab()` function to create a summary table from our shuffled decks, so that we can directly compare our expected values with our simulated observed values (spend a minute playing with the `crosstab()` function to make sure you understand what it does):

```
nrep = 1000;
```

```

simcounts = zeros(2,5);
simerrs = zeros(nrep,1);

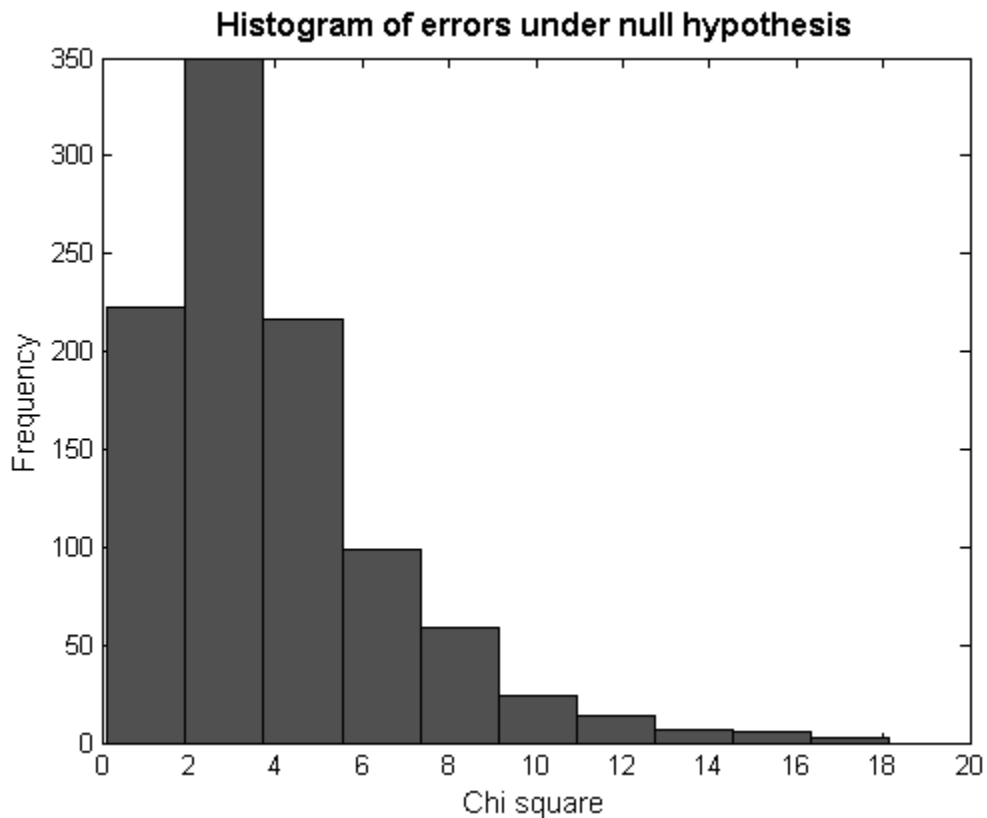
for i=1:nrep
    simgf = randsample(gf, totalsum);
    simgm = randsample(gm,totalsum);
    simcounts = crosstab(simgf,simgm);
    simerrs(i) = sum(sum(((simcounts-E).^2)./E));
end

hist(simerrs)

```

Notice that the last line simply computes the chi-squared error metric defined above, and stores it in the vector `simerrs` for the *i*th experiment.

When I ran this, I got the following distribution:



All we have left to do is compute the value of the chi-squared statistic computed from our data and compare it to the above distribution:

```
> realerr = sum(sum(((realcounts-E).^2)./E));
realerr =
    2.5430
```

This value is obviously in the heart of the distribution, which means that there is no evidence of an interaction between medium and growth factor. In other words, it looks like the two variables are operating completely independent of one another. Although we don't really need to compute a p-value because the outcome is so obvious, we can do so by counting the number of times our simulated experiments produced a valued of chi-squared equal to our greater than the one we computed from our data (as we did in the card game example):

```
> sum(simerrrs>=realerr)/nrep
ans =
    0.6400
```

which is obviously “not significant”!

Summary

In this chapter, we saw how to analyze data that consist of counts of the number of instances in qualitative categories. The canonical analysis of this kind of data is the “chi-squared test.” The chi-squared test is problematic, however, when the number of counts in any one category (or combination of two categories) is low, because the normal approximation to the binomial distribution breaks down. We can overcome this limitation by simply simulating the sampling distribution of the chi-squared error metric (or a similar error metric of our own devising) under the null hypothesis.

Summary of MATLAB used in this chapter

Exercises:

Problem 1

The following data were generated from a study in which subjects were classified as either depressed or not depressed. A neutral questionnaire was also administered, and the experimenters determined whether the subjects' modal answer was yes, don't know (?), or no.

	Yes	?	No
Depressed	65	5	10
Not depressed	13	5	2

Analyze these data using the last example in the chapter as a guide.

Compare the p-value to what you obtain using a standard chi-squared test. What do you think accounts for the discrepancy?

Problem 2

It has long been thought that birds use magnetic fields to navigate during migration. As a test, a migratory bird was placed in a square cage in a windowless room. The floor of the cage was divided into four triangular shaped quarters (picture and “X” drawn from corner to corner in the cage). Each quarter section of the floor was actually a big switch that recorded the number of times the bird hopped in that quarter. If the bird could sense magnetic fields, its migratory instinct might manifest itself in increased hopping on the North (or South) side of the cage. If, on the other hand, it could not sense magnetic fields, then it would probably hop around randomly.

Devise a test to determine whether the counts of hops in the four quadrants are consistent with the notion that the bird is hopping about randomly.

Back to our bird, the data showed that it had 72, 122, 65, and 97 hops in the E, N, W, and S quadrants respectively. Run your test on these data (part of the test should be knowledge of the distribution of your statistic, which you should be able to get easily via a Monte Carlo simulation).

Problem 3

In a more recent experiment, the researchers designed a better apparatus and then ran the bird in 3 conditions. In the new apparatus, the bird’s cage was surrounded by Helmholtz coils (which create magnetic fields under an experimenter’s control) and then the whole thing was enclosed in a copper foil tent (which shielded the cage from all electromagnetic fields originating outside the cage).

In the first condition, a simulated magnetic field was created that mimicked the earth’s (North pole toward geographic North). In the second, a simulated field was created that was the same strength as the earth’s, but with the North magnetic pole in the direction of

geographic East. In the final experiment, the coils were switched completely off, so no magnetic field was present in the cage.

If our bird cannot sense magnetic fields, then the pattern of hops across direction should have nothing to do with the state of the magnetic field, in other words, the variables “quadrant” and “state of magnetic field” should be independent.

	N	S	E	W
Natural	109	101	84	75
Perpendicular	81	65	88	91
Off	88	78	90	69

Modify and implement the test you devised above in this situation. The extension of the test itself should be straightforward, and you should be able to compute the expected values by referring to the last example in the chapter.

Problem 4

A recent poll of 1000 people about an upcoming gubernatorial race predicted the incumbent receiving 40.1% of the vote with the remainder going to a mix of other candidates. Is this consistent with people supporting or not supporting the incumbent based on a coin flip?

With a sample of 1000 people, what is the smallest departure from 50/50 that would make you predict that the incumbent (say) would win?

<p>Use Monte Carlo methods to answer these questions, then compare what you obtained with an analysis based on the binomial distribution.</p>

<p>Tip: An easy way to get a single coin flip in MATLAB is `round(rand(1))`. An easy way to get the number of heads in a kiloflip is `sum(round(rand(1000, 1)))`. </p>

Appendix

In the chapter, we gave an intuitive explanation of why the expected value, E , appears in the denominator of chi-squared, which is that it scales the statistic so that it is independent of the raw number of counts considered. Obviously, if you are counting 10,000 things, then fluctuations on the order of 10 are to be expected by chance (e.g., 5,010 people preferring one political candidate vs. 4,090 people preferring the other), whereas if you are counting 20 things, then fluctuations of 10 would represent substantial departures from chance (e.g. 10 people each preferring Chevy pickups to Ford pickups, vs. 20 people preferring Chevy to 0 preferring Ford).

The more curious students in my classes, however, have not been completely satisfied with this explanation. Specifically, they have wondered why “ E ”, which is equal to np by the normal approximation to the binomial, and not npq ($= np(1-p)$) which is the variance of the binomial distribution by the normal approximation.

The general proof is complex, but the proof for two groups is relatively straightforward. The key is that the expression for chi-squared (with E in the denominator) is getting *summed* to form the test statistic.

For two groups, we have

$$\begin{aligned} \sum_{k=1}^2 \frac{(O_k - E_k)^2}{E_k} &\approx \sum_{k=1}^2 \frac{(O_k - np_k)^2}{np_k} = \frac{(O_1 - np_1)^2}{np_1} + \frac{(O_2 - np_2)^2}{np_2} \\ &= \frac{(O_1 - np_1)^2}{np_1} + \frac{((n - O_1) - n(1 - p_1))^2}{np_2} \\ &= \frac{(O_1 - np_1)^2}{np_1} + \frac{(-O_1 + np_1)^2}{np_2} \\ &= \frac{p_2(O_1 - np_1)^2 + p_1(-O_1 + np_1)^2}{np_1 p_2} \\ &= \frac{(1 - p_1)(O_1 - np_1)(O_1 - np_1) + p_1(-O_1 + np_1)(-O_1 + np_1)}{np_1 p_2} \\ &= \frac{(O_1 - O_1 p_1 - np_1 + np_1^2)(O_1 - np_1) + (-O_1 p_1 + np_1^2)(-O_1 + np_1)}{np_1 p_2} \end{aligned}$$

Expanding the numerator and canceling terms gives:

$$= \frac{(O_1^2 - 2O_1np_1 + n^2p_1^2)}{np_1p_2}$$

$$= \frac{(O_1 - np_1)^2}{np_1(1 - p_1)}$$

From the normal approximation to the binomial,

$$\mu \approx np \text{ and } \sigma^2 \approx np(1 - p)$$

So the above becomes

$$X_1^2 \approx \frac{(O_1 - \mu)^2}{\sigma^2} = z^2$$

And is thus distributed as chi-squared (on one degree of freedom) by definition, Q.E.D.!

Programming tip – rolling an n-sided die

One way to take a uniformly distributed random number, scale it up to cover the range you want, and then round it up to the next highest integer. To roll a 3 sided die 100 times:
`> ceiling(3*rand(100,1))`

Another strategy is to define the faces of the die with a vector, and then sample from that vector

```
> mydie = [1,2,3];
> randsample(mydie, 100, true)
```

A nice thing about this method is that you can easily make unfair die if you need to:

```
> eachsideprob = [.499, .002, .499]; # a vector of probabilities
> simflips = randsample(mydie, 100, true, eachsideprob) # 100 coin flips
```

Usually you are ultimately after the number of “heads” or “door #3s” after a large number of trials. Let “1” be a head, “3” be a tail, and “2” be the edge of a coin:

```
> sum(simflips == 2) # how many times did the coin land on edge?
```