Open in Colab

Definition

The binomial distribution is a discrete probability distribution that describes the probability of obtaining exactly k successes out of n Bernoulli trials, which are individual binary outcomes that each have a success probability of p. For example, if you flip a coin 10 times, what is the probability that it will come up heads 0, 1, 2, ... 10 times?

Mathematically, this distribution defines the probability of obtaining *k* successes in *n* trials given a constant probability of success *p* on each trial, which is defined as:

[1] $P(X = k\ successes) = \binom{n}{k} p^k (1-p)^{n-k}$

for *k* = 0, 1, 2, ... , *n*, and where

$\binom{n}{k} = \frac{n!}{k!(n-k)!}$

is called the "binomial coefficient" and is pronounced "n-choose-k"

Getting Started with Code

Matlab code is found in the NGG Statistics GitHub Repository under "Probability Distributions/Binomial.m".

Python code is included below. First run the code cell just below to make sure all of the required Python modules are loaded, then you can run the other cell(s).

```
import numpy as np
import random as rnd
import collections
import matplotlib.pyplot as plt
import time
import scipy.stats as st

from scipy.stats import bernoulli, binom, poisson, chi2
from IPython.display import clear_output
from operator import itemgetter
from statsmodels.stats import proportion

from numpy import matlib
```

Tutorial

[ ] ↳ *4 cells hidden*

Neuroscience Example: Quantal Release

From: Johnson, E.W., and Wernig, A. (1971). The binomial nature of transmitter release at the crayfish neuromuscular junction. J Physiol 218, 757-767.

Classic studies by Bernard Katz and colleagues indicated that chemical neurotransmitters are released from presynaptic nerve terminals in discrete quanta, or packets, with a relatively constant release probability. A straightforward implication of this idea is that the release statistics (i.e., the distribution of the number of quanta that are actually released in response to a given event like an action potential reaching the presynaptic nerve terminal) should follow a binomial distribution. As stated by Johnson and Wernig:

"If the average probability of release is constant, then it follows from the hypothesis that the relative frequency of 0, 1, 2, 3, . . . quantal releases in a series of trials is given by the successive terms in the binomial expansion $(p+q)^n$, where *q*=1−*p* is the average probability that a quantum will not be released. In a given trial, the probability that *x* quanta will be released (*px*) is given by the binomial term" [i.e., Eq. 1, above, but substitute *x* for *k*].

In other words, if there are *n* available quanta in a presynaptic terminal, and each is released with probability *p*, then the number of quanta that are actually released should follow a binomial distribution with parameters *n* and *p*.

Answers to the exercises below will be found here after the due date.

Exercise 1

Assume that there are 10 quanta available in a nerve terminal, and for a given release event each is released with a probability of 0.2. For one such event, what is the probability that 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, or 10 quanta will be released?

```
p=0.2
n= 10
k = np.arange(11)
probs = binom.pmf(k,n,p)

for A, B in zip(k, probs):
  print(f'k={A:2d}: p={B:.2f}')
```

```
k= 0: p=0.11
k= 1: p=0.27
k= 2: p=0.30
k= 3: p=0.20
k= 4: p=0.09
k= 5: p=0.03
k= 6: p=0.01
k= 7: p=0.00
k= 8: p=0.00
k= 9: p=0.00
k=10: p=0.00
```

Exercise 2

Let's say you know that a given nerve terminal contains exactly 14 quanta available for release. You have read in the literature that the release probability of these quanta is low, say 0.1. To assess whether this value is reasonable, you run a simple experiment: activate the nerve and measure the number of quanta that are released. The result is 8 quanta. What is the probability that you would get this result (8 quanta) if the true probability of release really was 0.1? What about if the true release probability was much higher; say, 0.7? What about for each decile of release probability (0.1, 0.2, ... 1.0)? Which value of release probability did you determine to be the most probable, given your measurement?

Note: here you are computing a likelihood function: a function describing how the value of the conditional probability *p*(*data* | *parameters*) changes when you hold your data fixed to the value(s) you measured and vary the value(s) of the parameter(s) of, in this case, the binomial distribution. Because you are varying the parameters and not the data, the values of the function are not expected to sum to one (e.g., you can have numerous parameters that have a very high probability of producing the given data) and thus this function is not a probability distribution (see here for an extended discussion). The maximum value of this function is called the maximum likelihood

```
n= 10
k= 8

ps = np.arange(0.1, 1.0, 0.1) #probability range
probs = [binom.pmf(k,n,x) for x in ps]

for A, B in zip(ps, probs):
    print(f'p={A:.1f}: binomial probability={B:.4f}')
```

```
p=0.1: binomial probability=0.0000
p=0.2: binomial probability=0.0001
p=0.3: binomial probability=0.0014
p=0.4: binomial probability=0.0106
p=0.5: binomial probability=0.0439
p=0.6: binomial probability=0.1209
p=0.7: binomial probability=0.2335
p=0.8: binomial probability=0.3020
p=0.9: binomial probability=0.1937
```

Exercise 3

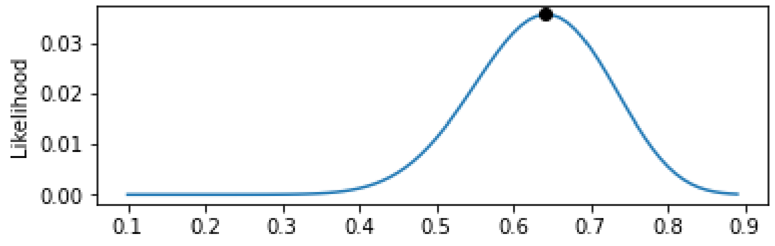
Not feeling convinced by your single experiment (good scientist!), you repeat it under identical conditions. This time you measure 5 quanta that were released. Your sample size has now doubled, to two measurements. You now want to take into account both measurements when you assess the likelihoods of different possible values of the underlying release probability. To do so, assume that the two measurements in this sample are independent of one another; that is, the value of each result had no bearing on the other. In this case, the total likelihood is simply the product of the likelihoods associated with each separate measurement. It is also typical to compute the logarithm of each likelihood and take their sum, which is often more convenient. What are the values of the total likelihood and total log-likelihood in this example, if we assume that the true release probability is 0.1?

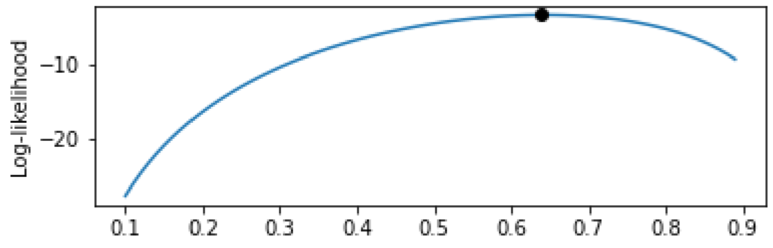
Of course, knowing those values of the likelihood and log-likelihood is not particularly useful until you can compare them to the values computed for other possible values for the release probability, so you can determine which value of release probability is most likely, given the data. Therefore, compute the full likelihood and log-likelihood functions using deciles of release probability between 0 and 1. What is the maximum value? Can you improve your estimate by computing the functions at a higher resolution? How does the estimate improve as you increase the sample size?

```
n1= 14 #Experiment 1 quanta value
k1= 8  #Measured release experiment 1 value
n2= 14 #Experiment 2 quanta value
k2= 10 #Measured release experiment 1 value
p= 0.1

prob1 = binom.pmf(k1,n1,p) # Probabilty of obtaining data 1 (k1) given n1, prelease
prob2 = binom.pmf(k2,n2,p) # Probabilty of obtaining data 2 (k2) given n2, prelease

total_probability = prob1 * prob2
total_log_probability = np.log(prob1) + np.log(prob2)
print(f'total probability = {total_probability:.3}, total log probability = {total_log_probability:.3f}')

#The likelihood and log-likelihood function
ps = np.arange(0.1, 0.9, 0.01)

#The binomial distribution of each
```

```
probs = binom.pmf(matlib.repmat([k1, k2], ps.size, 1),
                  matlib.repmat([n1, n2], ps.size, 1),
                  matlib.repmat(np.asmatrix(ps).T, 1, 2))

#Plot of the likelihood function
plt.subplot(211)
plt.ylabel('Likelihood')
likelihood_function = np.prod(probs,1)          # Compute the product for each row
plt.plot(ps, likelihood_function)               # Plot it
max_likelihood = np.amax(likelihood_function)   #  Get the maximum likelihood
plt.plot(ps[likelihood_function==max_likelihood], max_likelihood, 'ko')
plt.show()

#Plot of the log-likelihood function
plt.subplot(212)
plt.ylabel('Log-likelihood')
log_likelihood_function = np.sum(np.log(probs),1); # Compute the sum for each row
plt.plot(ps, log_likelihood_function)               # Plot it
max_log_likelihood = np.amax(log_likelihood_function)      # Get the maximum likelihood
plt.plot(ps[log_likelihood_function==max_log_likelihood], max_log_likelihood, 'ko')
plt.show()
```

```
total probability = 1.05e-12, total log probability = -27.584
```





Exercise 4

You keep going and conduct 100 separate experiments and end up with these results:

Measured releases	Count
0	0
1	0
2	3
4	10
5	19
6	26
7	16
8	16
9	5
10	5
11	0
12	0
13	0
14	0

What is the most likely value of p (which we typically refer to as $\hat{p}$, which is pronounced as "p-hat" and represents the maximum-likelihood estimate of a parameter in the population given our sample with a resolution of 0.01?

BONUS: Use a fitting procedure to find $\hat{p}$.

```
counts = [0, 0, 3, 10, 19, 26, 16, 16, 5, 5, 0, 0, 0, 0, 0] # The experimental outcomes
n = len(counts)-1                # Number of available quanta in each experiment
ks = np.arange(n+1)              # Possible values of k
ps = np.arange(0, 1.01, 0.01)    # Possible values of release probability, as a column vector

# Compute the value of the binomial distribution for each possible value of k, n, p. Make a matrix in which:
#     - columns correspond to different values of p
#     - rows correspond to different values of k
```

```
probs = binom.pmf(matlib.repmat(ks, ps.size, 1), n, matlib.repmat(np.asmatrix(ps).T, 1, ks.size))

# Make a matrix of outcomes (in rows) that are repeated along the columns so we can use them to compute likelihood
counts_matrix = matlib.repmat(counts, ps.size, 1)

# Compute likelihood function, which takes the product of all likelihoods associated with each measured outcome.
likelihood_fcn = np.prod(probs ** counts_matrix, axis=1)
p_hat_from_liklihood = ps[np.argmax(likelihood_fcn)]

# Compute log-likelihood function, which takes the sum of all log-likelihoods associated with each measured outcom
probs[probs<0.0001]=0.0001
log_likelihood_fcn = np.sum(np.log(probs) * counts_matrix, axis=1)
p_hat_from_log_likelihood = ps[np.argmax(log_likelihood_fcn)]

# Compute p directly from the empirical data
p_empirical = np.sum(counts*ks)/(np.sum(counts)*n)

# Show the results
print(f'p = {p_hat_from_liklihood:.3f} from likelihood, {p_hat_from_log_likelihood:.3f} from log-likelihood, {p_em
```

```
    p = 0.380 from likelihood, 0.380 from log-likelihood, 0.382 directly
```

Exercise 5

Let's say that you have run an exhaustive set of experiments on this synapse and have determined that the true release probability is 0.3 (within some very small tolerance). Now you want to test whether changing the temperature of the preparation affects the release probability. So you change the temperature, perform the experiment, and measure 7 quantal events for the same 14 available quanta. Compute $\hat{p}$. Standard statistical inference now asks the question, what is the probability that you would have obtained that measurement given a Null Hypothesis of no effect? In this case, no effect corresponds to an unchanged value of the true release probability (i.e., its value remained at 0.3 even with the temperature change). What is the probability that you would have gotten that measurement if your Null Hypothesis were true? Can you conclude that temperature had an effect?

```
n = 14                 # Number of available quanta
k = 7                  # Measured number of released quanta
```

```
p_hat = k/n           # Compute maximum-likelihood value of p
p_null = 0.3;         #  Null hypothesis p

# Get p-value for one-sided test that you would have gotten k or more successes in n trials for the given null hyp
p_value = st.binom_test(k, n, p_null, alternative='greater')


# Print result. Note that p>0.05, so cannot rule out that we would have gotten this measurement by chance under th
print(f'p value for the null hypothesis that p={p_null:.1f} = {p_value:.4f}')

     p value for the null hypothesis that p=0.3 = 0.0933
```

Double-click (or enter) to edit

Bonus Exercise

These data are reproduced from Tables 1 and 2 of Johnson and Wernig. Note that *nx* indicates the number of trials ("release events") with *x* measured events. Each row is a separate experiment (don't worry about the ordering, which here is reproduced as it is in the paper).

Temp (°C)	n0	n1	n2	n3	n4	n5+
4.0	615	206	33	2	0	0
3.5	604	339	94	11	2	0
0.0	332	126	21	1	0	0
2.0	573	443	154	28	2	0
6.5	172	176	89	12	1	0
3.0	80	224	200	32	4	0

You can probably already see that some experiments resulted in primarily zero or one events per trial (e.g., the top experiment), whereas others tended to have more events (e.g., the bottom experiment).

Do the quantal event numbers follow a binomial distribution? If so, then that would lend support to the idea that the events are generated by a process that can be characterized in terms of the number of available quanta and the average probability of release of each quantum.

Provide a qualitative answer to this question by comparing the empirical distributions (see the data table, above) to the predicted binomial distribution. To get the predicted distribution (i.e., the probabilities associated with each value of *k* from Eq. 1, given the release probability p and the number of available quanta *n*), you need to determine both *k* and *n* from the empirical measurements. To determine *p*, use Eq. 5 in the paper: $p = 1 - \frac{\sigma^2}{m}$

where σ^2 is the variance and *m* is the mean of the trial-by-trial counts that can be determined from the data. To determine *n*, compute *m*/*p* (because *m* = *n* x *p*).

As an added bonus, also include the predicted Poisson distribution, which can be computed using only the mean number of events (*m*, described above).

Additional Resources

Working with the binomial distribution in Matlab, R, and Python.

Credits