Project #2

Dylan Robertson
PHSX 815

May 11, 2023

Abstract

This project compared two hypotheses and attempted to distinguish between the two. The Null Hypothesis assumed that the rate parameter λ for a Poisson Distribution was constant, while the Alternative Hypothesis assumed that the rate parameter λ was distributed according to a Gamma Distribution. Data was generated for each hypothesis with $N_{meas} = 1,000$ measurements per experiment, and $N_{exp} = 10,000$ experiments total. A 95% Confidence Level was used for the Hypothesis comparison, which was done using the Log Likelihood Ratio. The power to distinguish between the two hypotheses was found to be 39.4%. Reasons for the low power of test will be discussed.

1 Null Hypothesis

The null hypothesis assumes that the rate parameter λ in a Poisson Distribution is constant. The Poisson Distribution is,

$$P(X|\lambda) = \frac{\lambda^X e^{-\lambda}}{X!}, \tag{1}$$

where λ is a positive, real number, representing the average rate that an event occurs at, and X is a whole number that represents the number of events that occurred within some fixed time interval.

The Poisson Distribution can be seen for various values of λ in Fig. 1. As λ gets sufficiently large, the distribution approaches that of a normal distribution, as predicted by the Central Limit Theorem.

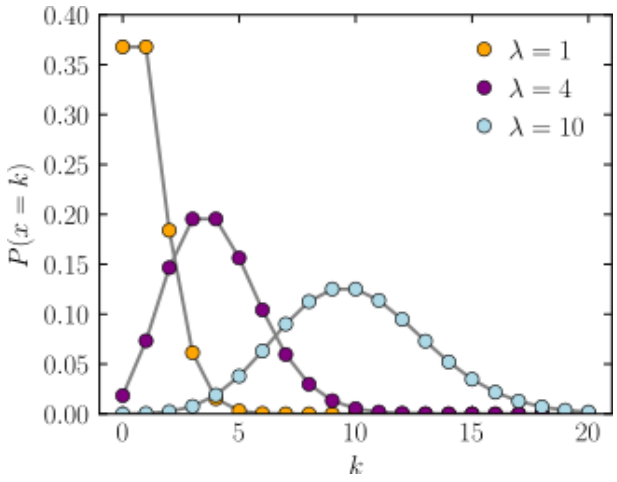

Figure 1: Poisson distributions with various values of λ. The expected peak in the distribution (the mean) occurs at the value of λ.

The likelihood of the a distribution is found by Baye's Theorem.

$$P(\lambda|X) \approx \prod_{i=1}^{N_{meas}} P(X_i|\lambda) = \prod_{i=1}^{N_{meas}} \frac{\lambda^{X_i} e^{-\lambda}}{X_i!} \tag{2}$$

where X_i is each individual measurement made in a single experiment and N_{meas} is the total number of measurements in the given experiment.

A more useful quantity is the logarithm of the likelihood, as it allows us to turn the product into a summation using properties of logarithmic functions. Going through the calculation, and using the Stirling approximation for the factorial results in the following expression.

$$ln(P(\lambda|X)) = \sum_{i=1}^{N_{meas}} [x_i ln(\lambda) - \frac{1}{2} ln(2\pi x_i) - x_i ln(x_i) + x_i - \lambda] \tag{3}$$

Which is the expression needed to calculate the likelihood of the Null Hypothesis given a sample data set.

For the simulation, random measurements from the Poisson Distribution were done using a standard function from the numpy library. The rate parameter was chosen to be $\lambda = 5.0$, the number of measurements per experiment was $N_{meas} = 1,000$, and the number of experiments was $N_{exp} = 10,000$. The data for the Null Hypothesis can be seen in Fig. 2.

```
#Poisson Distribution
    def Poisson(self, lamb):
        return np.random.poisson(lam=lamb, size=1)
```

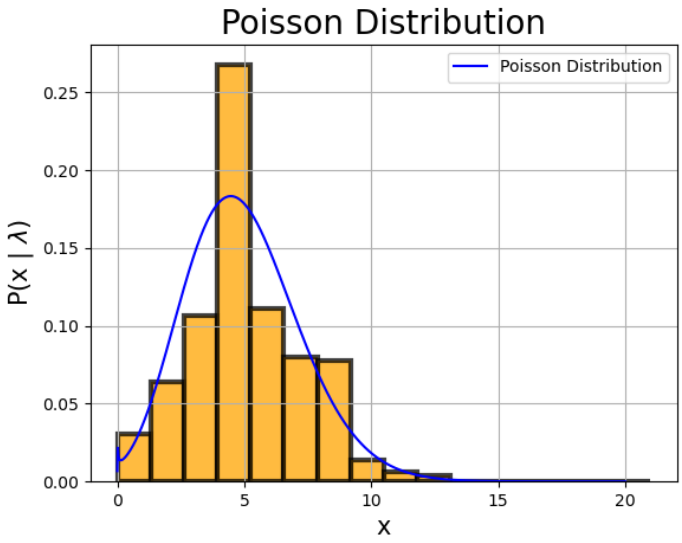

Figure 2: The data generated by the Null Hypothesis assuming a constant rate parameter $\lambda = 5.0$. The curve in blue is the actual distribution.

2 Alternative Hypothesis

The Alternative Hypothesis assumed that the rate parameter λ was distributed according to a Gamma Distribution. The Gamma Distribution is,

$$P(\lambda|\alpha,\beta) = \frac{\beta^{\alpha}}{\Gamma(\alpha)}\lambda^{\alpha-1}e^{-\beta\lambda}, \tag{4}$$

where α is a positive, real number representing the shape of the distribution, β is a positive, real number representing the width of the distribution, and $\Gamma(\alpha)$ is the Gamma function.

Fig. 3 shows the Gamma Distribution for various values of α and β. As the value $\frac{\alpha}{\beta}$ gets sufficiently large, the distribution approaches a normal distribution as predicted by the Central Limit Theorem.

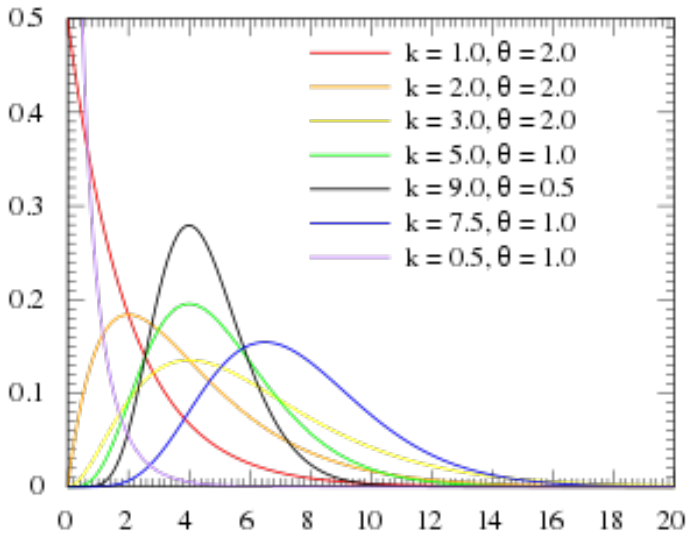

Figure 3: Gamma distributions with various values of $\alpha = k$ and $\beta = \frac{1}{\theta}$. The expected peak in the distribution (the mean) occurs at the value of $\frac{\alpha}{\beta}$.

The simulation for the Alternative Hypothesis worked as follows. The parameters for the Gamma Distribution were chosen such that the most likely λ value remains the same as the Null Hypothesis: $\alpha = 6.0$, and $\beta = 1.2$ For each new experiment, a new value of the rate parameter λ was generated from this Gamma Distribution. Then a data set X was generated from a Poisson Distribution for that value of λ. The number of measurements per experiment and the number of experiments remains the same as that for the Null Hypothesis.

Fig. 4 shows the distribution of λ values according to the Gamma Distribution, and Fig. 5 shows the data generated under this multi-step hypothesis.

Random measurements of λ from the Gamma Distribution were done using a standard function from the numpy library.

```
#Gamma Distribution
    def Gamma(self, alpha, beta):
```

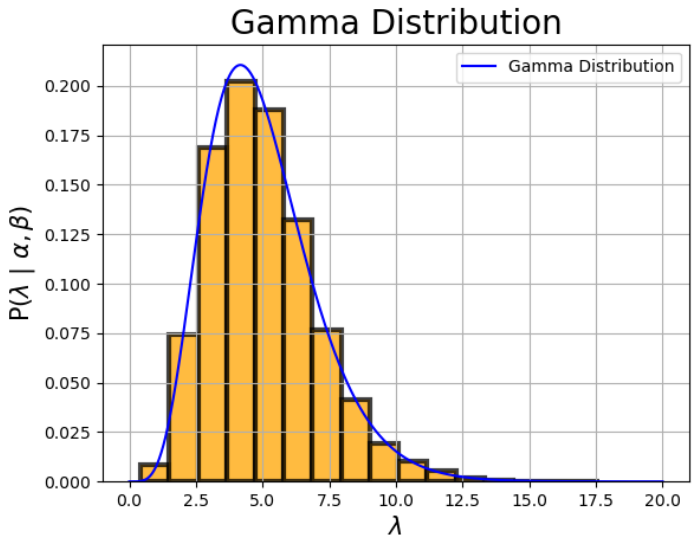

Figure 4: The distribution of λ values according to a Gamma distribution with parameters $\alpha = 6.0$ and $\beta = 1.2$. The expected peak in the distribution (the mean) occurs at the value of $\lambda = \frac{\alpha}{\beta} = 5.0$. The curve in blue is the actual distribution.

```
        #both alpha and beta must be positive
        k = alpha
        theta = 1/beta

        return np.random.gamma(shape = k, scale = theta, size = 1)
```

3 Hypothesis Comparison

Hypothesis Comparison was done using the Log Likelihood Ratio (LLR).

$$LLR = ln(\frac{P(X|H_0)}{P(X|H_1)}) = ln(P(X|H_0)) - ln(P(X|H_1)) \tag{5}$$

where H_0 is the Null Hypothesis described by the parameter λ, and H_1 is the Alternative Hypothesis described by the parameters α and β.

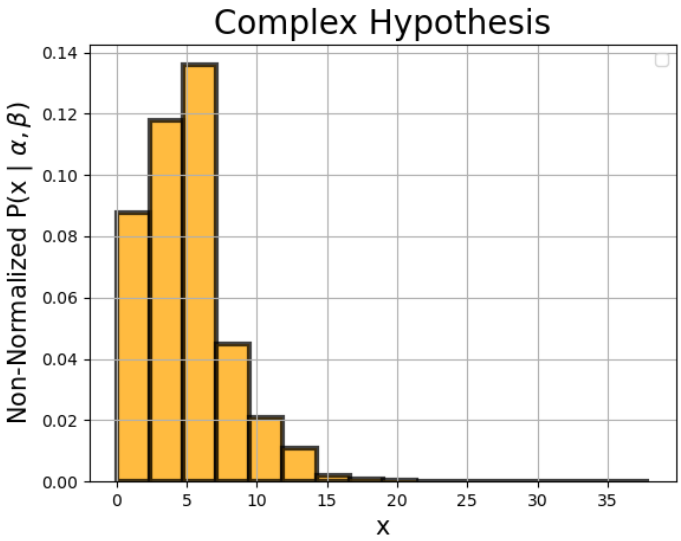

Figure 5: The data generated under the Alternative Hypothesis where the rate parameter of a Poisson Distribution λ was distributed according to a Gamma Distribution.

The log likelihood of a data set for the Null Hypothesis is described succinctly by Eqn. 3. The Alternative Hypothesis is more complex due to it's multi-step nature. The probability distribution shown in Fig. 5 is described by the following equation.

$$P(X|\alpha,\beta) = \int P(X|\lambda)P(\lambda|\alpha,\beta)d\lambda \tag{6}$$

where $P(X|\lambda)$ is the Poisson Distribution, and $P(\lambda|\alpha,\beta)$ is the Gamma Distribution, and all λ values are summed over.

The likelihood is once again given by Baye's Theorem shown in Eqn. 2. While this likelihood can be calculated analytically, we instead took a numerical approach using the histogram shown in Fig. 5. While useful to visualize the data under the Alternative Hypothesis, the histogram was really plotted in order to save two lists. The first list, n_{save}, contains the height of each bin. The second list, $bins_{save}$, has the location of each bin edge. Once the histogram has been normalized (by ensuring that the sum of all of the bin heights is equal to one), the height of each bin is the probability of measuring that data point (number of events).

Overall, the log likelihood function for the complex hypothesis is as follows.

```
    #Log Likelihood for Complex Hypothesis
    logprob_min = np.log(1/len(hist_complex))

    def ComplexLikelihood(data): #takes in one experiment
        g = 0 #initialize value

        for d in data: #measurements in an experiment
            bin_current = 0

            #protect against going over the farthest right bin edge
            if d > bins_save[len(bins_save) - 1]:
                logprob = logprob_min

            #find what bin the measurement falls in, starting from
                the left
            else:
                while d > bins_save[bin_current + 1]:
                    bin_current += 1

                #protect against bins w/ no counts
                if n_save[bin_current] <= 0:
                    logprob = logprob_min

                else:
                    logprob = np.log(n_save[bin_current]) #once
                        normalized, height of the histogram =
                        probability of measurement

            g += logprob

        return g
```

Generating the Log Likelihood Ratio, as shown in Eqn. 5, for each Hypothesis requires that you compute the likelihood of each hypothesis for each data set. The code is shown below.

```
#Log Likelihood Ratios
```

```
LL_simple_simple = [] #Log Likelihood of the Simple Hypothesis
    using the data made from the Simple Hypothesis
LL_complex_simple = [] #Log Likelihood of the Complex
    Hypothesis using the data made from the Simple Hypothesis

LL_simple_complex = [] #Log Likelihood of the Simple Hypothesis
    using the data made from the Complex Hypothesis
LL_complex_complex = [] #Log Likelihood of the Complex
    Hypothesis using the data made from the Complex Hypothesis

LLR_simple = []
LLR_complex = []

#Log Likelihood of the Simple Hypothesis using the data made
    from the Simple Hypothesis
for exp in data_simple: #each experiment
    LL = LogLikelihoodPoisson(lamb, exp)
    LL_simple_simple.append(LL)

#Log Likelihood of the Complex Hypothesis using the data made
    from the Simple Hypothesis
for exp in data_simple: #each experiment
    LL = ComplexLikelihood(exp)
    LL_complex_simple.append(LL)

#Log Likelihood of the Simple Hypothesis using the data made
    from the Complex Hypothesis
for exp in data_complex: #each experiment
    LL = LogLikelihoodPoisson(lamb, exp)
    LL_simple_complex.append(LL)

#Log Likelihood of the Complex Hypothesis using the data made
    from the Complex Hypothesis
for exp in data_complex: #each experiment
    LL = ComplexLikelihood(exp)
    LL_complex_complex.append(LL)

#LLR for simple hypothesis
for i in range(len(LL_simple_simple)):
    LLR_simple.append(LL_simple_simple[i] -
```

```
        LL_complex_simple[i])

#LLR for complex hypothesis
for i in range(len(LL_simple_complex)):
    LLR_complex.append(LL_simple_complex[i] -
        LL_complex_complex[i])
```

The last portion of the code sorted the LLR lists in ascending order and then found the power of the test by finding the critical LLR value (determined by the confidence level) in both lists.

```
#Sort the LLRs
LLR_simple = Sorter.DefaultSort(LLR_simple)
LLR_complex = Sorter.DefaultSort(LLR_complex)

#Define Confidence Level and find critical LLR value
alpha_CL = 0.05 #Confidence Level = 95%

LLR_alpha_CL = LLR_simple[math.ceil(Nexp * alpha_CL)] #critical
    LLR value

#Find Beta and Power of Test
for i in range(len(LLR_complex)):
  if LLR_complex[i] >= LLR_alpha_CL:
      LLR_Beta = LLR_complex[i]
      LLR_Beta_Position = i
      break

Beta_CL = (len(LLR_complex) - LLR_Beta_Position)/Nexp #Beta_CL
    = percent of entries in LLR_complex above LLR_alpha_CL

Power = 1 - Beta_CL
```

4 Results

Results are summarized in Fig. 6. The power of the test to discern between the two hypothesis was found to be 39%, which is quite low for a number of reasons. The first is that the number of measurements per

experiment, N_{meas} was only $1,000$, which is quite low. Using a higher value for N_{meas} would improve the power of the test. The second reason is that the two hypothesis had the same Most Likely Estimate (MLE) for λ. Changing the parameters of the Alternative Hypothesis, α and β to create a different MLE for the value of λ would make the two hypotheses more distinguishable.

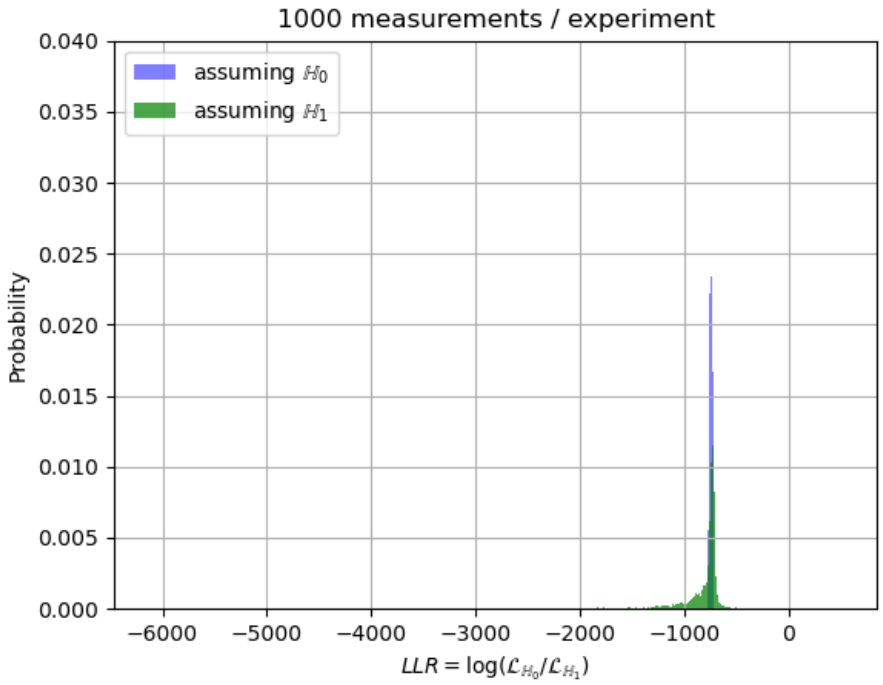

Figure 6: A graph of the log likelihood ratio for the two Hypotheses. The Null Hypothesis (blue) assumes a constant rate parameter λ, while the Alternative Hypothesis assumes that the rate parameter λ is distributed according to a Gamma Distribution. Parameter values were chosen such that both hypotheses had the same MLE for the value λ, which makes it hard to differentiate between the two hypotheses.

5 Summary

This paper described the process for how to use the Log Likelihood Ratio to distinguish between two hypotheses. The Null Hypothesis was quite simple, assuming a constant rate parameter of $\lambda = 5$ for the Poisson Distribution. The Alternative Hypothesis utilized a new rate parameter for each experiment, being randomly generated from a Gamma Distribution with $\alpha = 6.0$, and $\beta = 1.2$. Due to the similarity of the two hypotheses, the power of the test was quite low, found to be 39.4%.

Improving the power of test should be done by simulating more measurements per experiment (which my computer could not handle for the given number of experiments), and by choosing values of α and β such that the Maximum Likelihood Estimate of the most-likely rate parameter λ is distinct between the two hypotheses.

6 Hypothesis.py

The code file **Hypothesis.py** was used to do the generation of the data and the analysis of the data for this simulation. Since most of the analysis was shown in Section 3, only the data generation will be shown here. The entire code file can be viewed in the GitHub Repository.

```
#import packages
import math
import numpy as np
import matplotlib.pyplot as plt
import sys

#import Random class
sys.path.append(".")
import Random as rng

#import Sorting class
sys.path.append(".")
import MySort as mys

# main function
if __name__ == "__main__":
    # if the user includes the flag -h or --help print the options
```

```
251     if '-h' in sys.argv or '--help' in sys.argv:
252         print ("Usage: %s [options]" % sys.argv[0])
253         print (" options:")
254         print ("  --help(-h)        print options")
255         print ("  -seed [integer number] seed")
256         print ("  -Nmeas [integer number] number of measurements
257             per experiments")
258         print ("  -Nexp [integer number] number of experiments")
259         print ("  -lambda [float number] Parameter for the Null
260             Hypothesis")
261         print ("  -alpha [float number] Parameter for the
262             Alternative Hypothesis")
263         print ("  -beta [float number] Parameter for the
264             Alternative Hypothesis")
265         print
266         sys.exit(1)
267 
268     #Initialize
269     seed = 5555
270 
271     Nmeas = 1 #number of measurements per experiment
272     Nexp = 1 #number of experiments
273 
274     lamb = 1.0 #must be positive
275     alpha = 1.0 #must be positive
276     beta = 1.0 #must be positive
277 
278     #System Inputs
279     if '-seed' in sys.argv:
280         p = sys.argv.index('-seed')
281         seed = sys.argv[p+1]
282 
283     if '-Nmeas' in sys.argv:
284         p = sys.argv.index('-Nmeas')
285         ptemp = int(sys.argv[p+1])
286         Nmeas = ptemp
287 
288     if '-Nexp' in sys.argv:
289         p = sys.argv.index('-Nexp')
290         ptemp = int(sys.argv[p+1])
```

```
291         Nexp = ptemp
292
293     if '-lambda' in sys.argv:
294         p = sys.argv.index('-lambda')
295         ptemp = float(sys.argv[p+1])
296         lamb = ptemp
297
298     if '-alpha' in sys.argv:
299         p = sys.argv.index('-alpha')
300         ptemp = float(sys.argv[p+1])
301         alpha = ptemp
302
303     if '-beta' in sys.argv:
304         p = sys.argv.index('-beta')
305         ptemp = float(sys.argv[p+1])
306         beta = ptemp
307
308     #class instance of Random and Sorting class
309     random = rng.Random(seed)
310     Sorter = mys.MySort()
311
312     #initialize data
313     data_simple = [] #[[exp1], [exp2], ...] each experiment in the
314         simple hypothesis
315     data_graph = [] #[meas1, meas1, ...] used to plot data from
316         simple hypothesis
317
318     data_complex = [] #[[exp1], [exp2], ...] each experiment in the
319         complex hypothesis
320     hist_complex = [] #[meas1, meas1, ...] every measurement in the
321         complex hypothesis from all lambdas
322     lamb_graph = [] #[lamb1, lamb2, ...] used to plot the
323         distribution of lambdas
324
325     #Generate Data for Simple Hypothesis (fixed lambda)
326     for e in range(0, Nexp): #each experiment
327         data_exp_simple = [] #all measurements in a given experiment
328
329         for m in range(0, Nmeas): #each measurement
330           measurement_simple = float(random.Poisson(lamb))
```

```
        data_exp_simple.append(measurement_simple)
        data_graph.append(measurement_simple)

    data_simple.append(data_exp_simple)

#Generate Data for Complex Hypothesis (lambda comes from a
    Gamma Distribution)
for e in range(0, Nexp): #each experiment
    data_exp_complex = [] #all measurements in a given
        experiment

    lamb_complex = float(random.Gamma(alpha, beta)) #new lambda
        every experiment

    lamb_graph.append(lamb_complex)

    for m in range(0, Nmeas): #each measurement
      measurement_complex = float(random.Poisson(lamb_complex))
      hist_complex.append(measurement_complex)
      data_exp_complex.append(measurement_complex)

    data_complex.append(data_exp_complex)
```