Project #3

Dylan Robertson
PHSX 815

April 11, 2023

Abstract

The Gaussian Distribution is one of the most fundamental distributions in Statistics. This experiment ran simulations that estimated the value of the mean parameter based on data that was generated from a Gaussian Distribution with a mean of $\mu = 1.00$ and a standard deviation $\sigma = 1.00$. Each experiment had 25 measurements, and 1000 experiments were run in total. The mean value of experiment 1 was estimated to be $\mu = 1.08 \pm 0.20$, and the overall mean value from all experiments was estimated as $\mu = 0.99 \pm 0.21$.

1 Introduction

A Gaussian Distribution is the classic "bell-shaped" curve and is described below.

$$P(x|\mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-1}{2\sigma^2}(x-\mu)^2}, \tag{1}$$

where μ is the mean and σ is the standard deviation of the distribution. The mean describes where the peak of the distribution is, and the standard deviation affects the width of the distribution. As with any probability distribution, the factor out front is for normalization such that the integral over all space is equal to one.

This experiment first simulated random draws from a Gaussian Distribution with a mean $\mu = 1.00$ and a standard deviation $\sigma = 1.00$. There were $N_{meas} = 25$ measurements per experiment, and $N_{exp} = 1000$ experiments ran in total. The data was generated using **Gaussian.py** and was then graphed using **GaussianPlot.py**. The graph of this data, and the corresponding Gaussian curve can be seen in Fig. 1.

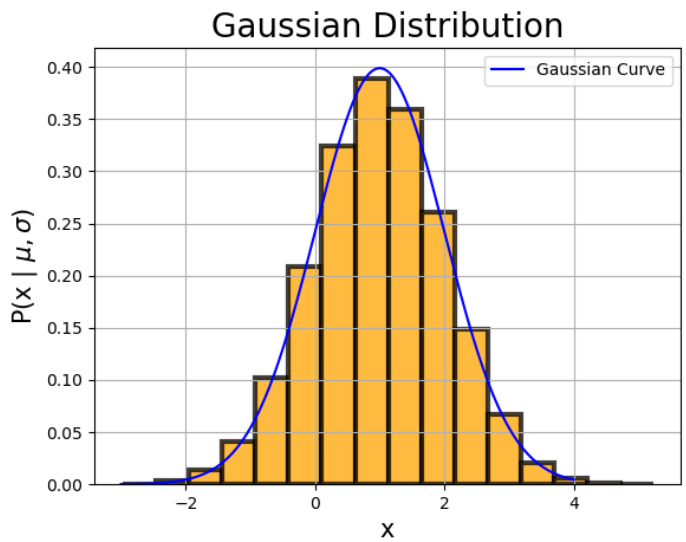

Figure 1: The data generated by a Gaussian with mean $\mu = 1.00$ and a standard deviation $\sigma = 1.00$. $N_{meas} = 25$ measurements/experiment and $N_{exp} = 100$ experiments. The expected peak in the data is when x equals the mean of the distribution.

A random measurement from the Gaussian distribution was done using a function for normal distributions in the numpy library.

```
#Gaussian Distribution
def Gaussian(self, mean, sigma):
    return np.random.normal(loc=mean, scale=sigma, size=1)
```

2 Theory

The above showed how to generate data from a given probability distribution. In our case, we are interested in estimating the mean parameter of the normal distribution given some data. Bayes Theorem states a useful relationship between the likelihood and the probability distribution.

$$P(\mu|x,\sigma) \approx \prod_{i=1}^{N_{meas}} P(x_i|\mu,\sigma) \tag{2}$$

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. If you go through the calculation you will find that the log likelihood for a Gaussian Distribution is the following.

$$ln(P(\mu|x,\sigma)) \approx -\frac{N_{meas}}{2}ln(2\pi) - \frac{N_{meas}}{2}ln(\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{N_{meas}}(x_i-\mu)^2, \tag{3}$$

where the first two terms are of no importance to us since they are constant and don't feature the mean μ in them.

The most common way to estimate a parameter is to find the value of the parameter that maximizes the likelihood (or the log likelihood). For a Gaussian distribution, this can be done analytically by taking partial derivatives and setting them equal to 0.

Maximizing the log likelihood with respect to the parameter μ yields,

$$\mu = \frac{1}{N_{meas}}\sum_{i=1}^{N_{meas}} x_i \tag{4}$$

which is the mean value of the experiment, as expected. This result will be used as a comparison for our numerical estimation.

3 Results

Results are summarized in Figs. 2 and 3. These graphs were created with **GaussianAnalysis.py**, which is shown in section 5.

The numerical estimation for the mean parameter from experiment one was $\mu = 1.08 \pm 0.20$, and the average estimation from all 1000 experiments was $\mu = 0.99 \pm 0.21$. The numerical maximization was done using the scipy.optimize library and minimizing the negative of the log likelihood function for a single experiment. The error bars on the numerical estimations were computed with a simple root finding algorithm that solved for the x-values when the log likelihood was equal to $\frac{1}{2}$.

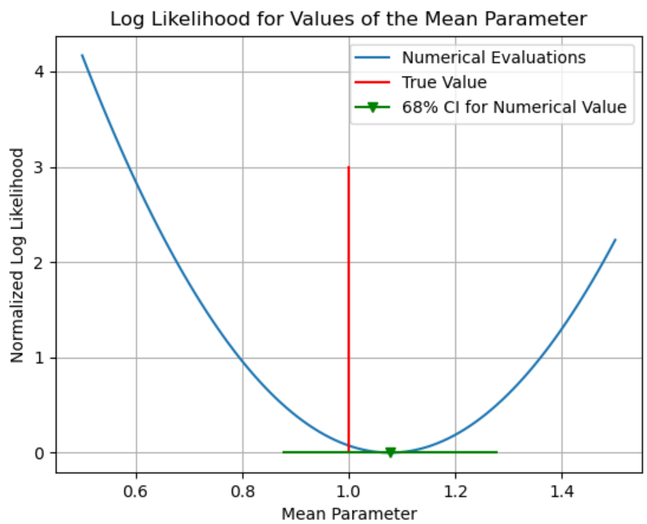

Figure 2: A graph of the log likelihood versus the mean parameter using the data from experiment 1 and comparing it to the analytical solution. The calculated mean value was $\mu = 1.08 \pm 0.20$ with 1-sigma error bars. The analytical solution was $\mu = 1.00$.

Figure 3: The numerical estimation of the mean parameter from all 1000 experiments. The peak of the distribution is the most likely value of the mean parameter and was found to be $\mu = 0.99 \pm 0.21$.

4 Summary

This simulation numerically estimated the mean parameter of a Gaussian Distribution using some data. This was done by finding the "most likely value" of the mean parameter μ by maximizing the log likelihood of our probability distribution using standard scipy libraries.

The mean value of experiment 1 was estimated to be $\mu = 1.08 \pm 0.20$, and the mean value from all experiments was estimated as $\mu = 0.99 \pm 0.21$, both of which are in agreement with the analytical solution.

While a normal distribution is quite a simple case, it is probably the most important one in statistics as all probability distributions asymptotically approach a normal distribution as the number of measurements gets very large. This result is known as the 'Central Limit Theorem'. Gaussian integrals are also quite easy to evaluate due to their rotational symmetry in higher dimensions, and are the mathematical basis for why the constant π seems to show up in problems that seemingly unrelated to circles.

5 GaussianAnalysis.py

Here are the key parts of the code file **GaussianAnalysis.py** used to do the analysis in this simulation.

The code reads in the data file generated by **Gaussian.py** and then finds a numerical estimate for the mean parameter μ for a single experiment, and then for all experiments. The numerical minimization/maximization was done using standard scipy libraries. Error on the mean parameter was found using a simple root finding algorithm. The error on the mean parameter σ_μ decreases as the number of measurements per experiment increases.

$$\sigma_\mu^2 = \frac{\sigma^2}{N_{meas}}, \tag{5}$$

where σ is the standard deviation of the distribution, and N_{meas} is the number of measurements per experiment.

```
#Analyze Data
    data_0 = [] # array of lists. Each list is an experiment.

    #Count Nmeas and Nexp
    with open(InputFile0) as ifile:
        for line in ifile: #Each line is a new experiment.
            lineVals = line.split() #All measurements in one experiment
            Nmeas = len(lineVals) #Each experiment has Nmeas measurements and is constant.

            data_exp = [] #gets reset each time.

            for v in lineVals: #all measurements in a single experiment
               val = float(v) #turns string into float
               data_exp.append(val)

            data_0.append(data_exp)

            Nexp += 1

#log likelihood for a single experiment
    current_exp = 0 #experiment 1
```

```

    def f(x):
        f = 0 #initialize value

        for d in data_0[current_exp]: #measurements in a single
            experiment
            f += -1/2 * np.log(2 * np.pi) - 1/2 * np.log(Sigma**2) -
                1/(2*Sigma**2) * (d - x)**2

        return -1 * f #need to return the negative in order to use
            minimization package.

    #analytical solution for maximum log likelihood
    x_max = Mean
    y_max = 0

    #numerical solution for single experiment (experiment 1)
    result = optimize.minimize_scalar(f)

    x_num = result.x #the numerical solution for the mean for a
        single experiment

    num_result = -1 * f(x_num) #normalization factor for graph

    #Log Likelihood curve
    x = np.linspace(0.5, 1.5, 100)
    y= []

#Initializtion for error analysis
    yerrlo = 0.5
    yerrhi = 0.5
    xerrlo = 0.
    xerrhi = 0.

    for i in range(len(x)):
        y.append(f(x[i]) + num_result)

        #find lower bound for 1 sigma error bar
        if x[i] < x_num:
            if np.abs(y[i] - 1./2.) < yerrlo:
```

```
                yerrlo = np.abs(y[i] - 1./2.)
                xerrlo = x[i]

        #find upper bound for 1 sigma error bar
        if x[i] > x_num:
            if np.abs(y[i] - 1./2.) < yerrhi:
                yerrhi = np.abs(y[i] - 1./2.)
                xerrhi = x[i]

    #Numerical error bars on x for a single experiment
    sig_num = (xerrhi - xerrlo)/2

    #calculate mean parameter and standard deviation of all experiments
    result_list = [] #list of all means

    Mean_exp = 0
    Mean_exp_squared = 0

    for exp in range(0, Nexp):
        current_exp = exp
        result = optimize.minimize_scalar(f) #result for each experiment
        result_list.append(result.x)

        Mean_exp += result.x
        Mean_exp_squared += result.x * result.x

    Mean_exp = Mean_exp/Nexp
    Mean_exp_squared = Mean_exp_squared/Nexp

    Sigma_exp = np.sqrt(Mean_exp_squared - Mean_exp**2)
```