# CS 206 Project Distribution Fit

Bruce Rushing

11 June 2020

## 1 Introduction

For this project, I was tasked to examine data samples and estimate a generating distribution and its parameters for those samples. For each sample, different distributions and their parameters were computed. Then I was tasked to compute statistics that could provide evidence for which model and parameters best accounted for the data. Several different statistics were chosen and computed for the end user to decide which model best predicted the data.

## 2 Methods

The problem presented to me was that my program would be given a vector of floats that are $n$ length, $x_0, x_1, ..., x_n \in \mathbb{R}$ and then it would have to guess the generating distribution $M$ with parameters $\theta$of that vector. The task thus is just to compute

$$P(\theta; M|x_0, ..., x_n) \tag{1}$$

Bayes theorem provides the means through which to calculate that value:

$$P(\theta; M|x_0, ..., x_n) = \frac{P(x_0, ..., x_n|\theta; M)P(\theta; M)}{P(x_0, ..., x_n)} \tag{2}$$

Because the assignment specified that the generating distribution was either uniform, normal, or exponential, i.e. $M \in \{Uniform, \mathcal{N}, Exp\}$, an approach relying upon maximum likelihoods could be used to estimate (1) by computing $P(x_0, ..., x_n|\theta; M)$. This method finds the best $\theta$ for each $M$ that best fits the data. In other words, the method of maximum likelihood computes $f(\theta)$ (Barber 2012) such that

$$f(\theta) = \arg\max P(x_0, ..., x_n|\theta; M) \tag{3}$$

To solve for (3), first note that likelihood, $P(x_0, ..., x_n|\theta; M)$, can be readily computed on assumption that each $x_i \in \{x_0, ..., x_n\}$ is conditionally independent of every $x_j \neq x_i$ such that $x_j \in \{x_0, ..., x_n\}$ on $\theta; M$. This means that

$$l(\theta) = P(x_0, ..., x_n|\theta; M) = \prod_{i=0}^{n} P(x_i|\theta; M) \tag{4}$$

Each individual conditional probability can be computed directly from the probability density function (PDF) of the given models $M$ with parameters $\theta$.[1] This allowed me to directly compute the maximum likelihood estimator (MLE) for each model.

## 2.1 Maximum Likelihood Estimators

Each maximum likelihood was computed analytically. The first case is for the uniform distribution. The PDF for the uniform, $X \sim Uniform(a, b)$ is given by

$$f(x) = \begin{cases} \frac{1}{b-a} & x \in [a, b] \\ 0 & otherwise \end{cases} \tag{5}$$

With this one can compute the MLE for the model $U$.

**Proposition 1.** *For data $\vec{x} = \langle x_0, ..., x_n \rangle$ where $x_0 \leq x_1 \leq x_2 \leq ... \leq x_n$, the MLE $f(a, b) = \arg\max P(\vec{x}|(a, b); U) = (x_0, x_n)$.*

*Proof.* I need to show that $f(a, b) = \arg\max P(\vec{x}|(a, b); U) = (x_0, x_n)$. First, I show that $a \geq x_0$ and $b \leq x_n$. Suppose $a > x_0$ for any $b$. Then $l(a, b) = 0$ by equation (5). By equation (4), $l(a, b) = 0$. Likewise for $b < x_n$. Second, I show that $a \not< x_0$ and $b \not> x_n$ in the MLE. Let $\delta = x_n - x_0$. Let $\gamma = b - a$ such that either $a < x_0$ or $b > x_n$ and $a \not> x_0$ and $b \not< x_n$. Then for any $i$, on the model where $b = x_n$ and $a = x_0$, $P(x_i|(x_0, x_n); U) = \frac{1}{\delta}$. Consequently, by equation (4),

$$l(x_0, x_n) = \frac{1}{\delta^n} \tag{6}$$

And for any model where $\gamma$ holds, we have

$$l(a, b) = \frac{1}{\gamma^n} \tag{7}$$

Note that because $\gamma > \delta$, equation (6) will be greater than (7). Therefore, $a \not< x_0$ and $b \not> x_n$. □

[1] Strictly speaking this is incorrect. What I am actually doing is taking conditional densities. The conditional probabilities would be given by the area under the PDF. But because individual data sets are probability zero events (the area under a point is 0), I default to densities. This fudge would be remedied by allowing for an alternative conditionalization instead of Bayesian conditionalization but given that there is no current consensus on which method to use, I default to standard practice in taking conditional densities to compute the relevant most probable *a posteriori*. I can do this fudge by substituting for the PDF an approximate probability mass function for individual conditional probabilities (Barber 2012, p. 18).

For computing the maximum likelihood for the normal, we note that the PDF for it, $X \sim \mathcal{N}(\mu, v)$, is given by

$$f(x) = \frac{1}{\sqrt{2\pi v}} e^{-\frac{1}{2v}(x-\mu)^2} \tag{8}$$

To make finding the MLE easier, I will use the loglikelihood. Plugging (8) into (4), we have

$$l(\theta) = \prod_{i=0}^{n} \frac{1}{\sqrt{2\pi v}} e^{-\frac{1}{2v}(x_i-\mu)^2} \tag{9}$$

Simplifying:

$$l(\theta) = \frac{1}{(\sqrt{2\pi v})^n} e^{-\frac{1}{2v}\sum_{i=0}^{n}(x_i-\mu)^2} \tag{10}$$

We can now compute the MLE for the normal.

**Proposition 2.** *For data $\vec{x} = \langle x_0, ..., x_n \rangle$, the MLE $f(\mu, v) = \arg\max P(\vec{x}|(\mu, v); \mathcal{N})$ for parameters $\mu$ and $v$ are:*

1. $\mu_{MLE} = \frac{\sum_{i=0}^{n} x_i}{n}$
2. $v_{MLE} = \frac{\sum_{i=0}^{n} (x_i-\mu)^2}{n}$

*Proof.* My strategy for finding MLE is to find the zeros for partial derivatives with respect to $\mu$ and $v$. To do this, I will solve for the partial derivatives of the loglikelihood. The results there will apply for the likelihood because $\log l(\theta) \propto l(\theta)$. Applying the natural logarithm to 10, this comes to

$$\log l(\theta) = \log[\frac{1}{(\sqrt{2\pi v})^n} e^{-\frac{1}{2v}\sum_{i=0}^{n}(x_i-\mu)^2}] \tag{11}$$

$$\log l(\theta) = \log(2\pi v)^{-\frac{n}{2}} + \log e^{-\frac{1}{2v}\sum_{i=0}^{n}(x_i-\mu)^2} \tag{12}$$

$$\log l(\theta) = -\frac{n}{2}\log 2\pi - \frac{n}{2}\log v - \frac{1}{2v}\sum_{i=0}^{n}(x_i-\mu)^2 \tag{13}$$

Call $\log l(\theta)$, $L(\theta)$. I now solve for the partial derivatives, starting with $\frac{\partial L}{\partial \mu}$:

$$\frac{\partial L}{\partial \mu} = 0 - 0 - (2)(-1)\frac{1}{2v}\sum_{i=0}^{n} x_i - \mu \tag{14}$$

$$\frac{\partial L}{\partial \mu} = \frac{1}{v}\sum_{i=0}^{n} x_i - \mu \tag{15}$$

Solving now for the zero of $\frac{\partial L}{\partial \mu}$:

$$0 = \frac{1}{v}\sum_{i=0}^{n} x_i - \mu \tag{16}$$

$$0 = \frac{1}{v}\sum_{i=0}^{n} (x_i) - (n\mu) \tag{17}$$

$$0 = \sum_{i=0}^{n} (x_i) - n\mu \tag{18}$$

$$n\mu = \sum_{i=0}^{n} x_i \tag{19}$$

$$\mu = \frac{\sum_{i=0}^{n} x_i}{n} \tag{20}$$

By Fermat's Theorem, equation (20) is a local maxima. A quick examination of the function also shows that (13) has a single maximum. Therefore, $\mu_{MLE} = \frac{\sum_{i=0}^{n} x_i}{n}$ or the sample mean, $\hat{\mu}$.

The strategy is the same for $v_{MLE}$. I first solve for the first partial derivative $\frac{\partial L}{\partial v}$:

$$\frac{\partial L}{\partial v} = 0 - \frac{n}{2}(\frac{1}{v}) - (-1)(\frac{1}{2v^2})\sum_{i=0}^{n}(x_i - \mu)^2 \tag{21}$$

$$\frac{\partial L}{\partial v} = -\frac{n}{2v} + \frac{1}{2v^2}\sum_{i=0}^{n}(x_i - \mu)^2 \tag{22}$$

Now, I solve for the 0:

$$0 = -\frac{n}{2v} + \frac{1}{2v^2}\sum_{i=0}^{n}(x_i - \mu)^2 \tag{23}$$

$$\frac{n}{2v} = \frac{1}{2v^2}\sum_{i=0}^{n}(x_i - \mu)^2 \tag{24}$$

$$nv = \sum_{i=0}^{n}(x_i - \mu)^2 \tag{25}$$

$$v = \frac{\sum_{i=0}^{n}(x_i - \mu)^2}{n} \tag{26}$$

Again, by Fermat's Theorem, this is a local maxima. This is just the sample variance, $\hat{v}$. □

Next I compute the MLE for the exponential distribution. The PDF for the exponential, $X \sim Exp(\lambda)$ is given by

$$f(x) = \lambda e^{-\lambda x} \tag{27}$$

Plugging this into (4), we have the following equation for the likelihood:

$$l(\theta) = \prod_{i=0}^{n} \lambda e^{-\lambda x_i} \tag{28}$$

Using this, we find the MLE:

**Proposition 3.** *For data $\vec{x} = \langle x_0, ..., x_n \rangle$, the MLE*

$$f(\lambda) = \arg\max P(\vec{x}|(\lambda); Exp) = \frac{n}{\sum_{i=0}^{n} x_i}$$

*Proof.* The strategy here is the same as in normal MLE. First, I define the loglikelihood and then find the zero for the first derivative of that likelihood. Taking the log of equation (28):

$$\log l(\theta) = \log[\prod_{i=0}^{n} \lambda e^{-\lambda x_i}] \tag{29}$$

$$\log l(\theta) = \sum_{i=0}^{n} \log \lambda e^{-\lambda x_i} \tag{30}$$

$$\log l(\theta) = \sum_{i=0}^{n} \log \lambda + \log e^{-\lambda x_i} \tag{31}$$

$$\log l(\theta) = n \log \lambda + \sum_{i=0}^{n} \log e^{-\lambda x_i} \tag{32}$$

$$\log l(\theta) = n \log \lambda - \lambda \sum_{i=0}^{n} x_i \tag{33}$$

Call $\log l(\theta)$, $L(\theta)$. Taking the first derivative:

$$L'(\theta) = \frac{n}{\lambda} - \sum_{i=0}^{n} x_i \tag{34}$$

Solving for the zero:

$$0 = \frac{n}{\lambda} - \sum_{i=0}^{n} x_i \tag{35}$$

$$\frac{n}{\lambda} = \sum_{i=0}^{n} x_i \tag{36}$$

$$\lambda = \frac{n}{\sum_{i=0}^{n} x_i} \tag{37}$$

This is just the reciprocal of the sample mean, $\lambda = \frac{1}{\bar{\mu}}$. By Fermat's theorem, this is a local maxima and an examination of $L(\theta)$ shows that there is only one maxima. So $\lambda_{MLE} = \frac{n}{\sum_{i=0}^{n} x_i}$. □

With these three estimators, I could compute the maximum likelihoods for each sample on each distribution. However, when running my experiments, I decided to also implement estimators using iterative methods instead of directly computing the MLE for each model. Those methods are the Newton-Raphson method and the gradient descent method.

## 2.2 Newton's Method on Estimators

The first method was to employ the Newton-Raphson method (Wasserman 2013, p. 143). This approach is an extension of Newton's method except instead of solving for the zeros of the likelihood function on each distribution, one solves for the zeros of the first derivative of the likelihood function. On each iteration, one then computes the value for $\theta$ by subtracting the previous iteration's value with the first partial derivatives of the likelihood function divided by the second partial derivatives of the likelihood function:

$$\theta_{k+1} = \theta_k - \frac{l'(\theta_k)}{l''(\theta_k)} \tag{38}$$

Note, it is important that the second partial derivatives $l''$ are non-zero. This will be important below for considering my results below.

Because the loglikelihood functions of the different distributions are better behaved, I used them in this method:

$$L_U(\theta) = -n \log(b - a) \tag{39}$$

$$L_{\mathcal{N}}(\theta) = -\frac{n}{2} \log 2\pi - \frac{n}{2} \log v - \frac{1}{2v} \sum_{i=0}^{n} (x_i - \mu)^2 \tag{40}$$

$$L_{Exp}(\theta) = n \log \lambda - \lambda \sum_{i=0}^{n} x_i \tag{41}$$

Taking the first derivatives of these functions, we have the following:

$$L'_U(\theta) = \begin{bmatrix} \frac{\partial L}{\partial a} = \frac{n}{b-a} \\ \frac{\partial L}{\partial b} = -\frac{n}{b-a} \end{bmatrix} \tag{42}$$

$$L'_{\mathcal{N}}(\theta) = \begin{bmatrix} \frac{\partial L}{\partial \mu} = \frac{1}{v}\sum_{i=0}^{n} x_i - \mu \\ \frac{\partial L}{\partial v} = -\frac{n}{2v} + \frac{1}{2v^2}\sum_{i=0}^{n}(x_i - \mu)^2 \end{bmatrix} \tag{43}$$

$$L'_{Exp}(\theta) = \begin{bmatrix} \frac{\partial L}{\partial \lambda} = \frac{n}{\lambda} - \sum_{i=0}^{n} x_i \end{bmatrix} \tag{44}$$

I then found the second partial derivatives:

$$L''_U(\theta) = \begin{bmatrix} \frac{\partial^2 L}{\partial a} = \frac{n}{(b-a)^2} \\ \frac{\partial^2 L}{\partial b} = \frac{n}{(b-a)^2} \end{bmatrix} \tag{45}$$

$$L''_{\mathcal{N}}(\theta) = \begin{bmatrix} \frac{\partial^2 L}{\partial \mu} = -\frac{n}{v} \\ \frac{\partial^2 L}{\partial v} = \frac{n}{v^2} - \frac{1}{v^3}\sum_{i=0}^{n}(x_i - \mu)^2 \end{bmatrix} \tag{46}$$

$$L''_{Exp}(\theta) = \begin{bmatrix} \frac{\partial^2 L}{\partial \lambda} = -\frac{n}{\lambda^2} \end{bmatrix} \tag{47}$$

Because Newton's method yields an approximate solution, it is important to have a stopping criteria (Bindel and Goodman 2009). I chose two stopping criteria:

1. **Max Iterations**: Stop after 1000 iterations.
2. **Difference In Change**: Computes the percentage change in the solution and checks to see if it is smaller than some $\epsilon = 0.00000001$:

$$\frac{\|x^{(k+1)} - x^{(k)}\|}{\|x^{(k+1)}\|} < \epsilon \tag{48}$$

I found that using both stopping criteria was important because sometimes the difference in change of the solution would not sink below the required amount in a timely manner. The reason for this is due to the behavior of the first and second partial derivative functions for the parameters, which I discuss now.

For both $L_U$ and $L_{\mathcal{N}}$, those partial derivatives and second partial derivaties are not well-behaved. When running my experiments on a number of test data samples, I would frequently find that the partial derivatives for $a$, $b$, and $v$ would grow massively, resulting in overflow errors for my program. Examining $L'_U(\theta)$ and $L'_{\mathcal{N}}(\theta)$ showed that the functions had asymptotes as well as very steep curves. This meant that the second partial derivatives either would not exist or would be close to vertical, leading to dividing by numbers bigger than could be represented by my machine.

Only $L_{Exp}$ had first and second derivatives that were well-enough behaved to allow this method to work well. However, it working needed the initial starting point, $\theta_0$, to be in the right place for the first and second derivatives to allow for the solution to be approximated. I found that $\theta_0 = [0.01]$ to work well enough. The current version of the code included in the appendix utilizes this method for estimating $\lambda$ on the exponential estimator.

## 2.3 Gradient Descent on Estimators

Because of the behavior of the first and second derivatives in the Newton-Raphson method, I also attempted the gradient descent method to see if it was any more effective. I only tested it on the normal maximum likelihood to see if it would work.

Gradient descent is a form of local search for finding the minima of function. It iteratively optimizes its target function through the gradient of that target function. The gradient of a function $f$ with parameters $x_0, x_1, ..., x_n$ ($\nabla f$) is a vector of that function's partial derivatives:

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_0} \\ \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix} \tag{49}$$

Each iteration is computed by subtracting the previous iteration with the gradient on that prior iteration multipled by a constant $\eta$:

$$x_{n+1} = x_n - \eta \nabla f(x_n) \tag{50}$$

Applying this successively, the minima of $f$ may be found. Because I wanted to find the maxima of the likelihood function, I did not reverse the direction of the gradient and instead just added it directly to the previous parameter values. This modified the previous equation to

$$x_{n+1} = x_n + \eta \nabla f(x_n) \tag{51}$$

This means that technically the method that was applied was the gradient ascent method.

Both the functions and stopping criteria employed in gradient descent were the same as in Newton's method. The functions that were optimized were the loglikelihoods from the previous section. The gradient in each case was the same as the first derivatives taken in the previous section with respect to each $L$. The stopping criteria utilized were difference in change and a maximum iterations in case the former failed.

I used a number of different values for $\eta$ between 0.1 and 0.01. The best success was found with smaller $\eta \approx 0.01$. The most iterations run was 10000.

Gradient descent proved less successul than the Newton-Raphson method. While I did not encounter any overflow errors, gradient descent applied to

the normal loglikelihood function rarely found the sample mean while variance would be off by multiple orders of magnitude from the sample variance. The likely explanation is that the loglikelihood function has multiple maxima and plateaus. This would cause the optimizer to be stuck. Larger values of $\eta$ can compensate for this but I found that this often led to values farther from the global maxima. This was likely due to the fact that the optimizer would just from one hill to another.

## 2.4 Statistics Computed

Recall that the task required in this assignment was to guess the generating distribution, $\theta; M$, from a sample of data, $x_0, ..., x_n$. This value is the posterior given by Bayes's theorem:

$$P(\theta; M|x_0, ..., x_n) = \frac{P(x_0, ..., x_n|\theta; M)P(\theta; M)}{P(x_0, ..., x_n)} \tag{52}$$

However, computing the posterior is often intractable over all models $M$ and parameters $\theta$ (Goodfellow, Bengio, and Courville 2016, p. 135). Due to this constraint, I computed a number of different statistics to allow the end user decide which distribution was most likely to have produced the sample. The statistics computed were the loglikelihoods, p-values computed on a Z-test, p-values computed on a K-S test, the posterior using MLE on a uniform prior, the Bayes's factor on MLE, and a confidence interval on the model with the highest posterior.

1. **Loglikelihoods**: The loglikelihoods were computed directly as given above by $L_U, L_{\mathcal{N}}, L_{Exp}$. Outputs were ranked according to the greatest loglikelihood.

2. **P-values with Z-test**: The p-value computed here is the probability that the test statistic would be more extreme than what was seen given the assumed generating distribution or $P(|T| > t|\theta; M)$. The test statistic used here was the Z-score, which tests the difference between the sample mean and mean of the assumed distribution $\theta; M$ normalized with respect to the sample standard deviation:

$$Z(\vec{x}) = \frac{\bar{x} - \mu}{\hat{\sigma}} \tag{53}$$

   The value of the Z-score can then be used to compute either a 1-sided or 2-sided p-value. Because assumed generating distribution are non-normal, I opted for a 2-sided p-value. This is computed by the following rule where CDF is the cumulative distribution function of $\theta; M$, i.e. $CDF_{\theta;M}$:

```
if Z-score > 0:
    p-value = 1 - CDF(Z-score) + CDF(- Z-score)
else:
    p-value = 1 + CDF(Z-score) - CDF(- Z-score)
```

It is important to note that the Z-test by normalizing the score through the sample standard deviation assumes that generating distribution is normal. As will be discussed below, this can lead to the computed p-value being biased with respect to models that are normally distributed.

3. **P-values with K-S Test**: Like the previous test, this computes the probability that the test statistic would have been more extreme than what was calculated on the assumped model. The test-statistic calculated here is the Kolmogorov-Smirnov statistic (K-S). It measures the biggest distance between an empirically fitted distribution $F_n$ and the assumed model:

$$D = \sup_x |F_n(x) - F(x)| \tag{54}$$

The empirically fitted distribution is constructed from a continuous, independent, and identically distributed ordered random variables $X_i$ that are plugged into an indicator function $I$ that returns 1 if its input is less than $x$ and 0 otherwise and then averaged (Wasserman 2013, p. 245):

$$F_n(x) = \frac{1}{n}\sum_{i=0}^{n} I(X_i) \tag{55}$$

One can think of the empirically fitted distribution as an approximate fit. I used Python's Scipy stats package to compute this p-value.

4. **Posterior on MLE**: This stastic restricts the posterior to just being calculated on the MLE estimates with a uniform prior. Each MLE estimate is assigned a probability drawn from $Uniform(0, 3)$ or $P(\theta; M) = \frac{1}{3}$. Then the posterior is computed using Bayes's theorem on the three MLE models.

An important limitation that was discovered when doing this was that the MLEs given in likelihoods could not be represented in any float available. This would lead to overflow errors. Consequently, I used loglikelihoods instead. This required that the probabilities computed were logarithmic probabilities, which I could then convert back to standard probabilities. The logarithmic form of Bayes's theorem for a $MLE$ given three total hypotheses $MLE_1, MLE_2, MLE_3$ is given by:

$$\begin{aligned}\log P(MLE|x_0, ..., x_n) = \log P(x_0, ..., x_n|MLE) + \log P(MLE) \\ -[\log P(x_0, ..., x_n|MLE - 1)P(MLE_1) + \log(1 \\ +e^{\log P(x_0,...,x_n|MLE_2)P(MLE_2) - P(x_0,...,x_n|MLE_1)P(MLE_1)} \\ +e^{\log P(x_0,...,x_n|MLE_3)P(MLE_3) - P(x_0,...,x_n|MLE_1)P(MLE_1)})]\end{aligned} \tag{56}$$

The assumption that a uniform prior is used means that this estimate will align with the maximum likelihood estimates ranked according to their respective likelihoods (Murphy 2012, pp. 70–72).

5. **Bayes Factor**: The Bayes factor on comparing individual models is the ratio between the likelihoods:

$$K = \frac{P(x_0, ..., x_n|\theta; M_1)}{P(x_0, ..., x_n|\theta; M_2)} \tag{57}$$

Typically, this ratio is computed by also integrating in the denominator with respect to all possible $\theta$ in model $M_2$. However, I decided to just utilize the MLE estimates on each model to compute their any given model's respective Bayes's factors to those models. The upshot is that the computed value is just a likelihood ratio test.

Note that the likelihoods computed are logarithmic for similar reasons for computing the posteriors. Each MLE estimate was given a Bayes Factor with respect to the other models. The way to read it in the tables is that each entry is the other models in numeric order. So the first entry will have [model ranked second, model ranked third] while the second entry will have [model ranked first, model ranked third] and likewise for the third entry.

6. **Confidence Interval**: A single confidence interval is computed on each run for the model with the highest likelihood/posterior. The confidence level, $c$, is set by the user on input out of allowed values 0.9, 0.95, 0.99, and 0.999. The confidence interval for each set of parameters is given by

$$C_{\hat{\theta}} = \hat{\theta} \pm z * \frac{1}{\sqrt{|L''(\hat{\theta})|}} \tag{58}$$

$L''$ is the second partial derivatives for each member of $\theta$ and I use it to compute the standard error. The value $z = |CDF^{-1}_{\mathcal{N}(0,1)}(\alpha)|$ where $\alpha = (1 - c)/2$.

# 3 Results

The results are broken down into those for required test data and additional data tests I ran with three additional distributions. The samples in this section had 10,000 data points and were taken from the from a normal, uniform, and exponential distribution. The data sets are included in submission as text files `d13.txt`, `d14.txt`, `d15.txt` for each distribution respectively.

## 3.1 Given Test Data

Table 1: Outputs on d1.txt. Confidence Interval at $c = 0.95$ is [0.3595729114440214, 0.3739496902557991] for $\lambda$ in $Exp(\lambda)$.

| Distribution | Log-likelihood | Z-test P-value | KS-test P-value | MLE Posterior | Bayes Factor (K) |
|---|---|---|---|---|---|
| Exponential [0.3667613] | -20030.44048936576 | 0.9981260802859298 | 0.36807302288319077 | 1.0 | [ 7653.8077283 12013.98428621] |
| Normal [2.72656902 2.70417551] | -27684.248217664303 | 1.0 | 1.2448908485395242e-213 | 9.85230...e-3325 | [-7653.8077283 4360.17655791] |
| Uniform [1.42147e-04 2.46419e+01] | -32044.424775573287 | 0.0 | 0.0 | 1.41861...e-4932 | [-12013.98428621 -4360.17655791] |

Table 2: Outputs on d2.txt. Confidence Interval at $c = 0.95$ is [2.7100346484318325, 2.7266253515681673] for $a$ and [3.133274648431833, 3.1498653515681676] for $b$ in $Uniform(a, b)$.

| Distribution | Log-likelihood | Z-test P-value | KS-test P-value | MLE Posterior | Bayes Factor (K) |
|---|---|---|---|---|---|
| Uniform [2.71833 3.14157] | 8598.158849461757 | 1.0 | 0.0 | 1.0 | [ 7881.66279607 29344.99658236] |
| Normal [2.92906651 0.12206068] | 716.4960533918621 | 1.0 | 1.2926467848999637e-26 | 1.08978...e-3423 | [-7881.66279607 21463.33378629] |
| Exponential [0.3414057] | -20746.83773289431 | 0.0 | 0.0 | 0.0 | [-29344.99658236 -21463.33378629] |



Table 3: Outputs on d3.txt. Confidence Interval at $c = 0.95$ is [2.641907065522342, 2.765400464393057] for $\mu$ and [3.1200908941338077, 3.1807085003864715] for $v$ in $\mathcal{N}(\mu, v)$.

| Distribution | Log-likelihood | Z-test P-value | KS-test P-value | MLE Posterior | Bayes Factor (K) |
|---|---|---|---|---|---|
| Normal [2.70365376 3.1503997 ] | -30679.03048238257 | 1.0 | 0.8351413598816311 | 1.0 | [ 724.66654471 187841.26468958] |
| Uniform [-8.36011 14.7523 ] | -31403.697027089074 | 0.0 | 0.0 | 1.91125...e-315 | [ -724.66654471 187116.59814487] |
| Exponential [0.36986985] | -218520.29517196323 | 0.0 | 0.0 | 0.0 | [-187841.26468958 -187116.59814487] |

Table 4: Outputs on d4.txt. Confidence Interval at $c = 0.95$ is [0.3396309232198685, 0.38451300410267103] $\lambda$ in $\mathcal{N}(\lambda)$.

| Distribution | Log-likelihood | Z-test P-value | KS-test P-value | MLE Posterior | Bayes Factor (K) |
|---|---|---|---|---|---|
| Exponential [0.36207196] | -2015.912292268661 | 0.999460153158885 | 0.65138160659024 | 1.0 | [784.07379121 943.8952388 ] |
| Normal [2.76188189 2.75036241] | -2799.9860834757033 | 0.9999999999999999 | 3.5465063184349558e-22 | 3.02746...e-341 | [-784.07379121 159.8214476 ] |
| Uniform [1.42147e-04 1.92944e+01] | -2959.8075310728664 | 0.0 | 0.0 | 1.17897...e-410 | [-943.8952388 -159.8214476] |

Table 5: Outputs on d5.txt. Confidence Interval at $c = 0.95$ is [2.692416388903711, 2.744843611096289] for $a$ and [3.1153563889037112, 3.167783611096289] for $b$ in $Uniform(a, b)$.

| Distribution | Log-likelihood | Z-test P-value | KS-test P-value | MLE Posterior | Bayes Factor (K) |
|---|---|---|---|---|---|
| Uniform [2.71863 3.14157] | 860.5249539682978 | 1.0 | 0.0 | 1.0 | [ 780.32973839 2934.56389935] |
| Normal [2.92717837 0.12021403] | 80.19521557697243 | 1.0 | 0.00132724381968842 | 1.27967...e-339 | [-780.32973839 2154.23416096] |
| Exponential [0.34162592] | -2074.0389453815387 | 0.0 | 0.0 | 3.42840...e-1275 | [-2934.56389935 -2154.23416096] |

Table 6: Outputs on d6.txt. Confidence Interval at $c = 0.95$ is [-8.00087678155711, -5.585663218442889] for $a$ and [11.483093218442889, 13.89830678155711] for $b$ in $Uniform(a, b)$.

| Distribution | Log-likelihood | Z-test P-value | KS-test P-value | MLE Posterior | Bayes Factor (K) |
|---|---|---|---|---|---|
| Uniform [-6.79327 12.6907 ] | -2969.5920762182077 | 0.5673573500410048 | 8.090887954417715e-100 | 1.0 | [ 163.32201982 20128.29808345] |
| Normal [2.66647823 3.24946176] | -3132.914096039013 | 1.0 | 0.854324730425742 | 1.17529...e-71 | [ -163.32201982 19964.97606363] |
| Exponential [0.3750265] | -23097.890159666185 | 1.3535839116229909e-12 | 0.0 | 0.0 | [-20128.29808345 -19964.97606363] |

Table 7: Outputs on d7.txt. Confidence Interval at $c = 0.95$ is [0.3071929655910226, 0.4569652172836074] for $\lambda$ in $Exp(\lambda)$.

| Distribution | Log-likelihood | Z-test P-value | KS-test P-value | MLE Posterior | Bayes Factor (K) |
|---|---|---|---|---|---|
| Exponential [0.38207909] | -196.21276461575272 | 0.9796624831847708 | 0.3965783676414079 | 1.0 | [83.49216669 86.89953028] |
| Normal [2.61725915 2.7460559 ] | -279.7049313055413 | 1.0 | 0.00013850642618806272 | 5.49303...e-37 | [-83.49216669 3.40736359] |
| Uniform [ 0.0204996 16.985 ] | -283.1122948937256 | 0.0 | 5.18130697030508e-36 | 1.81975...e-38 | [-86.89953028 -3.40736359] |

Table 8: Outputs on d8.txt. Confidence Interval at $c = 0.95$ is [2.636916750164251, 2.801283249835749] for $a$ and [3.056226750164251, 3.220593249835749] for $b$ in $Uniform(a, b)$.

| Distribution | Log-likelihood | Z-test P-value | KS-test P-value | MLE Posterior | Bayes Factor (K) |
|---|---|---|---|---|---|
| Uniform [2.7191 3.13841] | 86.91447758172131 | 1.0 | 1.010191373450513e-87 | 1.0 | [ 75.58047177 294.53477066] |
| Normal [2.9335196 0.11328564] | 11.334005809517844 | 1.0 | 0.41028398259053767 | 1.49905...e-33 | [-75.58047177 218.95429889] |
| Exponential [0.34088744] | -207.62029308000115 | 0.0 | 7.364216270920552e-88 | 1.21667...e-128 | [-294.53477066 -218.95429889] |

Table 9: Outputs on d9.txt. Confidence Interval at $c = 0.95$ is [-10.214949644678228, -3.371590355321772] for $a$ and [7.242920355321772, 14.086279644678228] for $b$ in $Uniform(a, b)$.

| Distribution | Log-likelihood | Z-test P-value | KS-test P-value | MLE Posterior | Bayes Factor (K) |
|---|---|---|---|---|---|
| Uniform [-6.79327 10.6646 ] | -285.9790549842543 | 0.6751914007739771 | 1.6670536842202188e-19 | 0.99999... | [ 24.40960181 2333.42556684] |
| Normal [2.49358449 3.20514023] | -310.3886567960105 | 1.0 | 0.9581311462793116 | 2.50636...e-11 | [ -24.40960181 2309.01596503] |
| Exponential [0.40102912] | -2619.40462182386 | 1.3929874771223538e-05 | 1.1495671776857861e-60 | 4.03787...e-1014 | [-2333.42556684 -2309.01596503] |

Table 10: Outputs on d10.txt. Confidence Interval at $c = 0.95$ is [0.14827466149179994, 0.631697585533519] for $\lambda$ in $Exp(\lambda)$.

| Distribution | Log-likelihood | Z-test P-value | KS-test P-value | MLE Posterior | Bayes Factor (K) |
|---|---|---|---|---|---|
| Exponential [0.38998612] | -19.416441212254206 | 0.973439173433814 | 0.8319877168460819 | 0.60512... | [0.44163435 4.65076727] |
| Uniform [0.186992 7.47192 ] | -19.858075562764334 | 0.7931802434304355 | 0.018599556909482003 | 0.38908... | [-0.44163435 4.20913292] |
| Normal [2.5641938 2.19114184] | -24.067208486795643 | 1.0 | 0.2530357752999524 | 0.00578... | [-4.65076727 -4.20913292] |

Table 11: Outputs on d11.txt. Confidence Interval at $c = 0.95$ is [2.51725204245966, 2.97350795754034] for $a$ and [2.8853220424596597, 3.34157795754034] for $b$ in $Uniform(a, b)$.

| **Distribution** | **Log-likelihood** | **Z-test P-value** | **KS-test P-value** | **MLE Posterior** | **Bayes Factor (K)** |
|---|---|---|---|---|---|
| Uniform [2.74538 3.11345] | 9.994821415109362 | 1.0 | 1.0663919057133492e-09 | 0.99983... | [ 8.7339254 30.7355034] |
| Normal [2.927264 0.11071746] | 1.260896018750001 | 1.0 | 0.8895929430669238 | 0.000161... | [-8.7339254 22.001578 ] |
| Exponential [0.34161593] | -20.740681983810887 | 2.348787830896981 2e-11 | 1.0605705564213617e-09 | 4.48405...e-14 | [-30.7355034 -22.001578 ] |



Table 12: Outputs on d12.txt. Confidence Interval at $c = 0.95$ is [-14.25523650987786, 0.6686965098778623] for $a$ and [-2.2158265098778624, 12.708106509877862] for $b$ in $Uniform(a, b)$.

| **Distribution** | **Log-likelihood** | **Z-test P-value** | **KS-test P-value** | **MLE Posterior** | **Bayes Factor (K)** |
|---|---|---|---|---|---|
| Uniform [-6.79327 5.24614] | -24.88185435357714 | 1.0 | 2.207999999999995e-07 | 0.99985...3 | [ 8.8705186 299.53914985] |
| Normal [1.0733089 3.62479875] | -33.752372950603196 | 1.0 | 0.6996056374868675 | 0.0001404... | [ -8.8705186 290.66863125] |
| Exponential [0.93169823] | -324.42100420340705 | 0.664116914041963 | 4.98355...e-09 | 8.160919712854889 5357e-131 | [-299.53914985 -290.66863125] |

Table 13: Outputs on true distribution $\mathcal{N}(1.8, 1.5)$. Confidence Interval at $c = 0.95$ is [1.766264537200207, 1.8243113270727358] for $\mu$ and [1.4623858108799754, 1.4992395215442205] for $v$ in $\mathcal{N}(\mu, v)$.

| Distribution | Log-likelihood | Z-test P-value | KS-test P-value | MLE Posterior | Bayes Factor (K) |
|---|---|---|---|---|---|
| Normal [1.79528793 1.48081267] | -18556.403842449236 | 1.0 | 0.7524434734532423 | 1.0 | [ 4703.07343017 115903.57097243] |
| Uniform [-3.34099054 6.89538625] | -23259.47727261518 | 0.6456846375709944 | 0.0 | 3.02803...e-2043 | [ -4703.07343017 111200.49754227] |
| Exponential [0.55701371] | -134459.97481488308 | 0.0 | 0.0 | 0.0 | [-115903.57097243 -111200.49754227] |

Table 14: Outputs on true distribution $Uniform(0.0, 4.0)$. Confidence Interval at $c = 0.95$ is [-0.07834808434667095, 0.07842535340360868] for $a$ and [3.9210477970799404, 4.07782123483022] for $b$ in $Uniform(a, b)$.

| Distribution | Log-likelihood | Z-test P-value | KS-test P-value | MLE Posterior | Bayes Factor (K) |
|---|---|---|---|---|---|
| Uniform [3.86345285e-05 3.99943452] | -13861.433200704181 | 0.9089823453239914 | 0.4789085382876377 | 1.0 | [1718.37630703 3089.52330019] |
| Normal [2.00390074 1.143762 ] | -15579.8095077379 | 1.0 | 3.2793709867596784e-28 | 5.23181...e-747 | [-1718.37630703 1371.14699316] |
| Exponential [0.49902671] | -16950.95650089728 | 0.9983350287687442 | 9.92515009744304e-221 | 1.72615...e-1342 | [-3089.52330019 -1371.14699316] |

Table 15: Outputs on true distribution $Exp(1.3)$. Confidence Interval at $c = 0.95$ is [1.2831280542109935, 1.334431274324253] for $\lambda$ in $Exp(\lambda)$.

| Distribution | Log-likelihood | Z-test P-value | KS-test P-value | MLE Posterior | Bayes Factor (K) |
|---|---|---|---|---|---|
| Exponential [1.30877966] | -7309.048509471736 | 0.9894106613100625 | 0.9728757950151803 | 1.0 | [ 4319.4419267 14583.60713162] |
| Normal [0.76407055 0.76097534] | -11628.490436171742 | 0.9999999999999999 | 2.0361577518490562e-216 | 1.23085...e-1876 | [-4319.4419267 10264.16520492] |
| Uniform [6.23766342e-05 8.92871556] | -21892.655641092813 | 0.0 | 0.0 | 1.41861...e-4932 | [-14583.60713162 -10264.16520492] |

# 4 Discussion

The performance of my program showed some interesting features in both the test data and my own generated samples. First, the program classified every exponential data sample with high credence and associated statistics. Second, the statistics computed on normal distributions with high variance were mixed with respect to that distribution and the uniform distribution. Third, the uniform prior used on the MLE posterior statistic hinders the program's classification. Fourth, the test statistics showed some degree in which model they would recommend.

My program showed an excellent record at classifying the exponential distributions. Each exponential data sample tended to have every statistic—p-value calculated with Z-test, p-value calculated with KS-test, MLE posterior, and Bayes's Factor—to signal that the generating distribution was exponential. Only one sample that the program judged to be exponential showed some indecisiveness, but even there all of the statistics tended to agree with the ranking of the different distributions (except the Z-test—more about that below). I believe this is likely due to the fact that the exponential shows behavior distinct from the other distributions, allowing the classifier to easily pick it out among the lot.

Unfortunately, my program showed less success at deciding between some sample uniform and normal distributions. Here the statistics computed often conflicted. Because the results were all ordered based on posteriors, I believe it is likely that multiple normal distributions were classified as uniform distributions. In those cases, the p-values calculated from KS-tests seems to offer a better recommendation. The reason for this confusion on the statistics is due to the fact that every normal distribution on those samples has a high variance. High variance normal distributions behave much like uniform distributions. This leads to the log-likelihoods being very close. Unfortunately, because they are logarithmic, any difference between them is likely to lead to big differences when computing the Bayes factor and posterior. This sort of information should be reflected in the prior when calculating the posterior.

The prior on the MLE posterior is uniform. Unfortunately, this is probably a poor choice. Because the likelihoods are very close, factoring additional information on the behavior of uniform and normal distributions from previous samples would be useful for the program. Successive data samples that demonstrate wide variance but different $a, b$ parameters on the uniform would add substantial credence to similar distributions being normal. This would change the prior from uniform to something more inclined to normal distributions over time. And because the log-likelihoods are close, that prior would then weigh and counterbalance the tendency to misclassify a high variance normal distribution as uniform.

Lastly, the statistics used in classifying generating distributions showed some degree of variance. In particular, the p-values varied widely based on the generating distribution. The Z-test repeadetely showed an inability to discriminate between the MLE normal distributions and other high likelihood candidates.

This is due to the fact that in calculating the Z-score, it is assumed that the behavior of the generating test statistic across the sample is normal. Consequently, the p-value on a normal MLE estimate will always be close to 1 since that estimate mean is always the sample mean and the error is distributed with the sample variance. What the p-values from the Z-test do tell one is when one should pay attention to the non-normal distributions as serious candidates.

## References


Barber, David (2012). *Bayesian reasoning and machine learning*. Cambridge University Press.

Bindel, David and Jonathan Goodman (2009). "Principles of scientific computing". In: *New York University, New York*.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep learning*. MIT press.

Murphy, Kevin P (2012). *Machine learning: a probabilistic perspective*. MIT press.

Wasserman, Larry (2013). *All of statistics: a concise course in statistical inference*. Springer Science & Business Media.


## Appendix: Code

```
#!/usr/bin/python3

"""
Distribution Fit Program

Bruce Rushing

6/10/2020

Estimates a generating distribution from a data sample.

Meant to be run in python3 or later.
"""


import argparse
import numpy as np
import math
from numpy import linalg as la
from scipy import stats

# True parameters
```

```
ALPHA = 0.0
BETA = 4.0
MU = 1.8
SIGMA = 1.5
LAMBDA = 1.3

# Constants
SMALL_N = 0.00000000000000000000000000000000000000000001
SIZE = 10000
ETA = 0.01
EPSILON = 0.00000001
MAX_ITER = 1000

# MLE Functions
def build_sample_uniform(length):
    return np.random.uniform(ALPHA, BETA, [length])

def build_sample_normal(length):
    return np.random.normal(MU,SIGMA,[length])

def build_sample_exponential(length):
    return np.random.exponential((1 / LAMBDA), [length])

def MLE_uniform(data):
    alpha = np.min(data)
    beta = np.max(data)
    return np.asarray([alpha, beta])

def MLE_normal(data):
    """
    #Code for running with gradient descent.
    x_0 = np.asarray([50.0, 50.0])
    l_nabla = lambda theta: loglikelihood_normal_fpp(data, theta)
    mle = gradient_ascent_func(x_0, l_nabla)
    return mle
    """
    mu = np.mean(data)
    sigma = np.sqrt(np.var(data))
    return np.asarray([mu, sigma])

def MLE_exp(data):
    """
    #Code for running based on analytic proof.
    lamb = 1 / np.mean(data)
    return np.asarray([lamb])
    """
```

```
    x_0 = np.asarray([0.01])
    l_p = lambda theta: loglikelihood_exp_fp(data, theta)
    l_pp = lambda theta: loglikelihood_exp_fpp(data, theta)

    mle = newtons_method_func(x_0, l_p, l_pp)
    return mle

def likelihood_uniform(x_bar, theta_bar):
    def pdf(x):
        if x >= alpha and x <= beta:
            return 1 / (beta - alpha)
        else:
            return 0

    alpha = theta_bar[0]
    beta = theta_bar[1]

    pdf_v = np.vectorize(pdf)
    return np.product(pdf_v(x_bar))

def likelihood_normal(x_bar, theta_bar):
    mu = theta_bar[0]
    sigma = theta_bar[1]

    pdf = lambda x: stats.norm.pdf(x, loc=mu, scale=sigma)
    pdf_v = np.vectorize(pdf)
    return np.product(pdf_v(x_bar))

def likelihood_exp(x_bar, theta_bar):
    lamb = 1 / theta_bar[0]
    min = np.min(x_bar)

    pdf = lambda x: stats.expon.pdf(x, loc=min, scale=lamb)
    pdf_v = np.vectorize(pdf)
    return np.product(pdf_v(x_bar))

def loglikelihood_uniform_func(x_bar, theta_bar):
    alpha = theta_bar[0]
    beta = theta_bar[1]

    def f(x):
        if x >= alpha and x <= beta:
            return np.log(1.0 / (beta - alpha))
        else:
            return np.log(SMALL_N)
```

```
    fv = np.vectorize(f)
    likelihood = np.sum(fv(x_bar))
    return likelihood

def loglikelihood_uniform_fpp(x_bar, theta_bar):
    alpha = theta_bar[0]
    beta = theta_bar[1]

    f = lambda x: 1 / ((beta - alpha) ** 2)
    fv = np.vectorize(f)
    likelihood = np.sum(fv(x_bar))
    return np.asarray([likelihood, likelihood])

def loglikelihood_normal_func(x_bar, theta_bar):
    mu = theta_bar[0]
    sigma = theta_bar[1]
    n = np.size(x_bar)

    f = lambda x: (x - mu) ** 2
    fv = np.vectorize(f)
    likelihood = ((-n / 2) * np.log(2 * np.pi * sigma)) - ((1 / (2 * sigma)) *
            np.sum(fv(x_bar)))
    return likelihood

def loglikelihood_normal_fp(x_bar, theta_bar):
    mu = theta_bar[0]
    sigma = theta_bar[1]
    n = np.size(x_bar)

    f = lambda x: x - mu
    g = lambda x: (x - mu) ** 2
    fv = np.vectorize(f)
    gv = np.vectorize(g)
    likelihood_mu = (1 / sigma) * np.sum(fv(x_bar))
    likelihood_sigma = (-n / (2 * sigma)) + ((1 / (2 * (sigma ** 2))) *
            (np.sum(gv(x_bar))))
    return np.asarray([likelihood_mu, likelihood_sigma])

def loglikelihood_normal_fpp(x_bar, theta_bar):
    mu = theta_bar[0]
    sigma = theta_bar[1]

    n = np.size(x_bar)
    f = lambda x: (x - mu) ** 2
    fv = np.vectorize(f)
    likelihood_mu = -n / (sigma ** 2)
```

```
    likelihood_sig = (n / (sigma ** 2)) + ((1 / (sigma ** 3)) * np.sum(fv(x_bar)))
    return np.asarray([likelihood_mu, likelihood_sig])

def loglikelihood_exp_func(x_bar, theta_bar):
    lamb = theta_bar[0]

    def f(x):
        if x >= 0.0:
            return np.log(lamb) - (lamb * x)
        else:
            return np.log(SMALL_N)

    fv = np.vectorize(f)
    likelihood = np.sum(fv(x_bar))
    return likelihood

def loglikelihood_exp_fp(x_bar, theta_bar):
    lamb = theta_bar[0]
    n = np.size(x_bar)
    sum_x = np.sum(x_bar)

    f = (n / lamb) - sum_x
    return np.asarray([f])

def loglikelihood_exp_fpp(x_bar, theta_bar):
    lamb = theta_bar[0]
    n = np.size(x_bar)
    f = ((-1 * n) / (lamb ** 2))
    return np.asarray([f])

def newtons_method_func(x_0, f, fp):

    def exit_func(steps, y):
        """
        Tells the algorithm to stop when percentage change between each iteration
        crosses below some epsilon.
        """

        y_p = y - (f(y) / fp(y))

        # Check to see if y_p = 0
        if y_p != 0.0:
            distance = la.norm((y_p - y)) / la.norm(y_p)
        else:
            distance = 0.0
```

```
        # Check the exit condition
        if distance < EPSILON:
            return False
        elif steps > MAX_ITER:
            return False
        else:
            return True

    x = np.asarray(x_0, dtype=np.double)
    count = 0

    while exit_func(count, x):
        count += 1
        # Check to see if fp == 0
        if np.all(fp(x) == 0.0) or x == 0.0:
            break

        # Compute next iteration
        #print('First derivative = ' + str(f(x)))
        #print('Second derivative = ' + str(fp(x)))
        x = x - (f(x) / fp(x))
        if np.all(f(x) == 0.0):
            break

    return x

def gradient_ascent_func(x_0, nabla_f):

    def exit_func(steps, y):
        """
        Tells the algorithm to stop when percentage change between each iteration
        crosses below some epsilon.
        """
        y_p = y + (ETA * nabla_f(y))

        # Check to see if y_p = 0
        if la.norm(y_p) != 0.0:
            distance = la.norm((y_p - y)) / la.norm(y_p)
        else:
            distance = 0.0

        # Check the exit condition
        if distance < EPSILON:
            return False
        elif steps > MAX_ITER:
            return False
```

```
        else:
            return True

    x = np.asarray(x_0, dtype=np.double)
    count = 0

    while exit_func(count, x):
        count += 1
        # Compute next iteration
        x = x + (ETA * nabla_f(x))

    return x

def confidence_interval_func(estimate, c, ldp):
    alpha = (1 - c) / 2
    z = np.abs(stats.norm.ppf(alpha))

    lower = estimate - (z * (1 / np.sqrt(np.abs(ldp(estimate)))))
    upper = estimate + (z * (1 / np.sqrt(np.abs(ldp(estimate)))))

    return [lower, upper]

def pvalue_func(sample, mean, null_cdf):
    z_score = (np.mean(sample) - mean) / stats.sem(sample)

    if z_score > 0:
        p = 1 - null_cdf(z_score) + null_cdf(-1 * z_score)
    else:
        p = 1 + null_cdf(z_score) - null_cdf(-1 * z_score)

    return p

def pvalue_ks_func(sample, null_cdf):
    ks, p_ks = stats.kstest(sample, null_cdf)
    return p_ks

def bayes_thm_func(likelihoods, prior, estimate=0):
    """
    Calculates a posterior given likelihood pdfs and a uniform prior on those pdfs.

    Calculations performed with logarithmic likelihoods and probabilities.
    Returned value is raised by e to return probabilities.
    """
    # Calculate evidence
    log_prior = np.log(prior, dtype=np.longdouble)
    p_numerator = np.longdouble(likelihoods[estimate] + log_prior)
```

```
    p_remainder = np.longdouble(1.0)
    p_evidence = p_numerator
    for i in range(np.size(likelihoods)):
        if i != estimate:
            p_exponent = np.longdouble(likelihoods[i]) + log_prior - p_numerator
            # If-statement to catch overflow error for what is representable using longdouble
            if p_exponent < 11356:
                p_remainder += np.exp(p_exponent, dtype=np.longdouble)
            else:
                p_remainder += np.exp(11354, dtype=np.longdouble)

    p_evidence += np.log(p_remainder)
    posterior = p_numerator - p_evidence
    return np.exp(posterior)

def bayes_factor_func(likelihoods, estimate=0):
    """
    Returns Bayes factors in logarithmic form.
    """
    factors = np.asarray([])
    for like in likelihoods:
        if like != likelihoods[estimate]:
            if like == 0.0:
                bayes_factor = float('inf')
            else:
                bayes_factor = likelihoods[estimate] - like
            factors = np.append(factors, bayes_factor)

    return factors

def compute_stats_func(sample, estimates, c):
    """
    Computes a series of statistics and outputs a table with those statistics.
    Returns a table.

    The statistics that are computed and displayed are:
    1. Likelihoods
    2. Z-Test P-values
    3. K-S P-values
    4. Posterior on Uniform Distribution
    5. Bayes Factor on Maximal Likelihood
    6. Confidence Interval on c for maximum likelihood parameters

    Inputs are the data sample, a dictionary of estimates, a confidence level
    (float), and a dictionary of second derivatives of likelihood function.
```

```
The structure of the dictionary estimates has to be the following:
key = distribution type, values = list with parameters, likelihood,
loglikelihood, cdf, ldp list.
"""
def key(x):
    return x[1][2]

# Sort the distributions on the loglikelihood
ranked_estimates = dict(sorted(estimates.items(), key=key, reverse=True))

# Collect likelihoods
prior = stats.uniform.pdf(1, loc=0, scale=3)
likelihoods = np.asarray([])
for e in ranked_estimates:
    likelihoods = np.append(likelihoods, ranked_estimates[e][2])

table = '{:<50}'.format('Distribution') + '{:<30}'.format('Log-Likelihood') + \
    '{:<30}'.format('Z-test P-value') + '{:<30}'.format('KS P-value') + \
    '{:<30}'.format('Posterior') + 'Average Bayes Factor\n'
tally = -100000000000000000.0
for e in ranked_estimates:
    # Notify the maximum item in the dictionary
    if ranked_estimates[e][2] > tally:
        tally = ranked_estimates[e][2]
        marker = e

    # Name the estimate
    table += '{:<50}'.format(e + ' ' + str(ranked_estimates[e][0]))

    # Add the likelihood
    table += '{:<30}'.format(str(ranked_estimates[e][2]))

    # Compute and add the p-value
    if e == 'Uniform':
        mean = (ranked_estimates[e][0][1] + ranked_estimates[e][0][0]) / 2
    elif e == 'Exponential':
        mean = np.min(sample) + (1 / ranked_estimates[e][0][0])
    else:
        mean = ranked_estimates[e][0][0]

    p_value = pvalue_func(sample, mean, ranked_estimates[e][3])
    table += '{:<30}'.format(str(p_value))

    # Compute KS p-value
    ks_p_value = pvalue_ks_func(sample, ranked_estimates[e][3])
    table += '{:<30}'.format(str(ks_p_value))
```

```
        # Posterior
        estimate = 0
        for i in range(np.size(likelihoods)):
            if ranked_estimates[e][2] == likelihoods[i]:
                estimate = i
        posterior = bayes_thm_func(likelihoods, prior, estimate=estimate)
        table += '{:<30}'.format(str(posterior))

        # Bayes Factor
        bayes_factor = bayes_factor_func(likelihoods, estimate=estimate)
        table += str(bayes_factor) + '\n'

    # Confidence Interval on c
    for i in range(np.size(ranked_estimates[marker][0])):
        par = ranked_estimates[marker][0][i]
        ldp = ranked_estimates[marker][4][i]
        c_interval = confidence_interval_func(par, c, ldp)
        table += 'Confidence Interval for the parameters is on ' +str(c) + \
                ' is ' + str(c_interval) + '\n'

    return table

def import_data_func(filename):

    with open(filename, 'r') as f:
        lines = f.readlines()

    lines = [x.strip() for x in lines]
    sample = [float(x) for x in lines]
    return np.asarray(sample)

if __name__ == "__main__":

    parser = argparse.ArgumentParser(description='Fits a distribution to a file of floats.')
    parser.add_argument('confidence', choices=[0.9, 0.95, 0.99, 0.999], type=float, default=0.95,
                        help='A confidence level for outputting confidence intervals')
    parser.add_argument('filename', type=str, help='A file name of floats.')

    args = parser.parse_args()

    filename = args.filename
    c = args.confidence
    sample1 = import_data_func(filename)

    sample_list = [sample1]
```

```
for s in sample_list:
    guess1 = MLE_uniform(s)
    guess2 = MLE_normal(s)
    guess3 = MLE_exp(s)

    # Calculate likelihoods and loglikelihoods
    like_unif = likelihood_uniform(s, guess1)
    like_norm = likelihood_normal(s, guess2)
    like_exp = likelihood_exp(s, guess3)
    loglike_unif = loglikelihood_uniform_func(s, guess1)
    loglike_norm = loglikelihood_normal_func(s, guess2)
    loglike_exp = loglikelihood_exp_func(s, guess3)

    # Build CDFs for likelihood distributions
    uniform_cdf = lambda x: stats.uniform.cdf(x, loc=guess1[0], scale=guess1[1])
    normal_cdf = lambda x: stats.norm.cdf(x, loc=guess2[0], scale=guess2[1])
    exp_cdf = lambda x: stats.expon.cdf(x, loc=np.min(s), scale=(1 / guess3[0]))

    # Build likelihood second partial deriviates for confidence intervals
    ldp_a = lambda a: loglikelihood_uniform_fpp(s, np.asarray([a, guess1[1]]))[0]
    ldp_b = lambda b: loglikelihood_uniform_fpp(s, np.asarray([guess1[0], b]))[1]
    ldp_mu = lambda mu: loglikelihood_normal_fpp(s, np.asarray([mu, guess2[1]]))[0]
    ldp_sig = lambda sig: loglikelihood_normal_fpp(s, np.asarray([guess2[0], sig]))[1]
    ldp_lam = lambda l: loglikelihood_exp_fpp(s, np.asarray([l]))[0]

    # Build dictionaries
    estimate_dict = {'Uniform': [guess1, like_unif, loglike_unif, uniform_cdf, [ldp_a, ldp_b]
                     'Normal': [guess2, like_norm, loglike_norm, normal_cdf, [ldp_mu, ldp_sig]
                     'Exponential': [guess3, like_exp, loglike_exp, exp_cdf, [ldp_lam]]}

    # Compute statistics and table
    table = compute_stats_func(s, estimate_dict, c)
    print(table)
```