Final Assignment - Statistical Methods for Computer Science - 18325745 - Ethan Monkhouse
Total Report Pages: **6 pages** | Total Appendix Pages: **2 Pages** | Total Report Word Count: **4085**

Question 1:
(a) i. To estimate the fraction of the population expected to test positive for COVID but have no significant symptoms, we must analyse all columns of the dataset. As the results are cumulative, we use the last row. The first column is our total population for the question, as the figure is: **2314355**. We then need to find the number of the tested population who did not show symptoms, yet were positive. We subtract the third column figure from the second column figure, **284947-194837 = 90110.** We then calculate the percentage in terms of the total population (**2314355**). This is achieved by calculating: *90110/2314355* = **0.038935. Answer: 0.038935** or **3.8935%**

ii. Since the third column is the people who tested positive and showed significant symptoms, we simply use this figure and put it over the total population. This is: *194837/2314355* = **0.084186. Answer: 0.084186** or **8.4186%**

We can confirm these fractions are accurate by totalling both (i) and (ii). The figure should total to the overall fraction of people tested. 0.038935 + 0.084186 = **0.123121** … 284947 / 2314355 = **0.123121.** From this, we can conclude that 12.31% of the population tested were positive. Of that, 3.89% showed no symptoms, and 8.42% showed significant symptoms. Suggesting that there is an overall greater chance of a person showing symptoms that not, when they have COVID.

(b) For this question, we are achieving a 95% confidence-interval. Firstly, for both parts of the previous question we have a binary random variable, either the amount of people testing positive with symptoms, or the amount of people testing positive without symptoms. There is no mention of mean and variance and we had previously calculated the probability of each event, 3.8935% and 8.4186%. We can calculate the standard deviation using the sampling distribution of a proportion:

$$\sigma_{p'} = \sqrt{\frac{p(1-p)}{n}} \qquad p' - Z_\alpha\sqrt{\frac{p'q'}{n}} \leq p \leq p' + Z_\alpha\sqrt{\frac{p'q'}{n}}$$

p is the probability of success, and n is the size of the sample, or in this case the population. We must then calculate the Z value for a 95% confidence interval. This can be extracted from the z-value table to be **1.96.** We then can use the above formula to calculate the confidence interval. As we don't know the exact value *p,* we use the estimate of *p* from the previous question, *p'*. *q'* is substituted for (1-p), which is the probability of *p'* not occurring.

(i) Event: test positive for COVID but have no significant symptoms.	(ii) Event: test positive and have significant symptoms.
Estimated probability: 0.038935 Z-Value: 1.96	Estimated probability: 0.084186 Z-Value: 1.96
Standard Deviation: sqrt((0.038935(1-0.038935))/2314355) = 1.2715E-4	Standard Deviation: sqrt((0.084186(1- 0.084186))/2314355) = 1.8252E-4
0.038935 - (1.96(1.2715E-4)) <= p <= 0.038935 + (1.96(1.2715E-4))	0.084186 - (1.96(1.8252E-4)) <= p <= 0.084186+ (1.96(1.8252E-4))
Upper Bound: 0.0391842 Lower Bound: 0.0386858	Upper Bound: 0.0845437 Lower Bound: 0.0838283
95% Confidence Interval [0.03868583, 0.0391842]	**95% Confidence Interval [0.0838283, 0.0845437]**

(c) In short, yes. The main factor we need to consider **skewness.** Skewness is usually described as the degree of symmetry of a dataset. When we assume that the people tested are drawn uniformly at random, we are assuming that the dataset is normally distributed. Logically, we are assuming that the dataset is perfectly symmetrical, and thus has a skewness of 0. If the dataset is not perfectly symmetrical, it will have either a positive or negative skew. This is usually caused by outliers, and in reference to the question, affects the mean of the data. When normally distributed, the mean, median, and mode are the same. If negatively skewed, the mean will be lower than the median and mode. This is the opposite when positively skewed. Consequently, if the data was not normally distributed we could experience inaccurate figures for the mean. Either over or under-estimating the proportion of the population with COVID. From a more practical perspective, consider the event in which the population of people who were tested were in a nursing home where the overall health conditions of persons is lower than the national average. This is not being drawn uniformly at random, and we can expect inflated results. Likely in terms of the number of people who show symptoms and have COVID. It is important that the population is truly random, relative to the conditions of the experiment/test. In this specific case, to ensure that the population is representative of the entire population, not just those in a specific age bracket or that have a specific health condition.

(d) In order to calculate the number of people who have COVID, we use marginalisation. Firstly, we take the number of people who have tested positive and show no symptoms are our base figure. We then calculate the number of people with a false positive. We take the number of positive cases (284947), and multiply it by the false positive rate of 0.01. For perspective purposes, we have calculated the overall proportion of the population this value represents. We then calculate the false negatives by subtracting the positive tests from the population and multiplying that value by the false negatives rate (0.1). Finally, to calculate the number of people who have COVID, we account for both false positives and false negatives. We take the figure that is the number of positive tests (90110), and subtract the false positive figure (2849) as they do not have COVID yet tested positive. We then add the false negatives figure (202941), as they do have COVID yet tested negative. We arrive at a value of **290193** people, and in proportion to the overall population - a proportion of **0.12538828** or **12.54%** is calculated as the proportion of people who have COVID. Workings below:

Number of people testing positive with COVID and have no significant symptoms	False positive calculation	False negative calculation	Number of people who have COVID but have no significant symptoms
284947-194837 = 90110 (*from dataset)* 90110/2314355 = **0.038935**	284947 * 0.01 = 2849 2849/2314355 = **0.0012310**	2314355 - 284947 = 2029408 2029408 * 0.10 = 202941 202941/2314355 = **0.087688**	90110 - Original Figure Less (2849) - False Positives + 202941 - False Negatives 90110 - 2849 + 202941 = 290193 290193/2314355 = **0.12538828**
3.89%	**0.123%**	**8.77%**	**12.54%**

(e) Similar to part (b), we are achieving a 95% confidence-interval for a binary random variable. There is no mention of mean and variance and we had previously calculated the probability that a person has COVID as 12.54%. We can calculate the standard deviation using the sampling distribution of a proportion:

$$\sigma_{p'} = \sqrt{\frac{p(1-p)}{n}} \qquad\qquad p' - Z_\alpha\sqrt{\frac{p'q'}{n}} \leq p \leq p' + Z_\alpha\sqrt{\frac{p'q'}{n}}$$

p is the probability of success, and n is the size of the sample, or in this case the population. We must then calculate the Z value for a 95% confidence interval. This can be extracted from the z-value table to be **1.96.** We then can use the above formula to calculate the confidence interval. As we don't know the exact value *p,* we use the estimate of *p* from the previous question, *p'*. *q'* is substituted for (1-p), which is the probability of *p'* not occurring.

Event: have COVID but have no significant symptoms.
Estimated probability: 0.12538828
Z-Value: 1.96
Standard Deviation: sqrt((0.12538828(1-0.12538828))/2314355) = 2.1768E-4

0.12538828 - (1.96(2.1768E-4)) <= p <= 0.12538828 + (1.96(2.1768E-4))

Upper Bound: 0.124961624
Lower Bound: 0.125814935

95% Confidence Interval [0.124961624, 0.125814935]

From the above interval, we can see that firstly the proportion of people who have COVID is greater than the proportion of people who have tested positive for COVID. This is largely due to the presence of false negatives, as well a reduction due to false positives. It can also be noticed that the 95% confidence interval is larger than part (b). This is due to an increase in the overall proportion, compared to part (b). It should be noted that in the event where the number of people tested was increased, we could expect a decrease in the deviation of the upper and lower bound of the confidence interval.

(f) To calculate the probability you have COVID given you test positive P(A|B), we must use Bayes Rule:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

P(B) is the probability that you test positive. We can calculate this by calculating the proportion of positive tests to the total population *(284947/2314355)*, giving a result of **0.12312156**. P(A) is the probability you have COVID, which was calculated in the previous part (e) as **0.12538828**. P(B|A) is the probability that you test positive with no symptoms given you have COVID. This can be calculated by subtracting the probability of a false negative from 1. *(1 - 0.1)*, giving **0.9**. We then substitute these figures into the Bayes formula above. P(A|B) = ((0.9)(0.12538828))/(0.1231215) = **0.91656938.** Thus the probability that you have COVID given you test positive for COVID with no symptoms is **0.91656938.**

(g)

```
dataset = importdata('/Users/ethanvictormonkhouse/final2021.php'); %import data
n = 100000; %run simulation n times
test_positive_p = max(dataset.data(:,2))/max(dataset.data(:,1)); %the probability of testing positive
test_positive_with_s_p = max(dataset.data(:,3))/max(dataset.data(:,2)); %the probability of having symptoms with a positive test
r=rand(1,n); %generate either a 0 or 1, n times
tested=(r<=(test_positive_p)); %label whether each value of r is positive or not
r=rand(1,sum(tested)); %for the amount of positive cases, generate a 0 or 1
test_positive=(r<(test_positive_with_s_p)); %label whether each value of r has symptoms or not
tp_sim_s_p = sum(test_positive)/n; %all positive with symptoms cases / population of simulation
tp_sim_ws_p = (length(test_positive) - sum(test_positive))/n; %all positive without symptoms cases / population of simulation
```

A stochastic simulation is a simulation of a system where variables change randomly based on their individual probabilities. The above snippet of code performs this simulation. Firstly, we set the population of the simulation. In this case it is 100,000 randomly generated events. We then set the probabilities of each event, that is: the probability a person tests positive, and the probability a person who has already tested positive, has significant symptoms. We can calculate the probability a person does not have symptoms with basic subtraction after. We generate a random variable which is either 0 or 1, n times. We then label whether each value of *r* is positive or not, based on the probability set *test_positive_p*. Then we need to simulate whether each of these positive cases have symptoms or not. We generate an array of random variables for the amount of positive cases (*sum(tested)*). Similar to before, we label each value of *r* has symptoms or not, based on the probability set *test_positive_with_s_p*. Finally, we can calculate our probability by totalling the array *test_positive* as the positive with symptoms have a value of 1, and then divide by the initial population n. For the probability of testing positive without symptoms, we take the length of *test_positive*, which is the total number of positive cases, and subtract the sum of the *test_positive.* This is essentially providing us with the number of *0* values in the array. We then divide by the initial population n.

(h)

	Test positive with symptoms	**Test positive without symptoms**	**Test positive (total)**
Initial Calculation (part a)	0.084186	0.038935	0.123121
Simulation (n=100000)	0.083580	0.039110	0.122690
Difference (initial - simulation)	0.000606	-0.000175	0.000431

In the table above is the results of the simulation when run where n=100000. We can see that the results are very similar to that of the initial calculation of part (a) of this question. We notice that less people tested positive with symptoms in the simulation. Conversely, more people tested positive without symptoms in the simulation. However, the total number of people who tested positive out of the entire population was less in the simulation than with the initial calculation. As much as these are interesting insights, the difference in figures between the simulation and initial calculation is negligible, and can only be used to confirm that our simulation is accurately developed and correct.

Question 2:

(a)

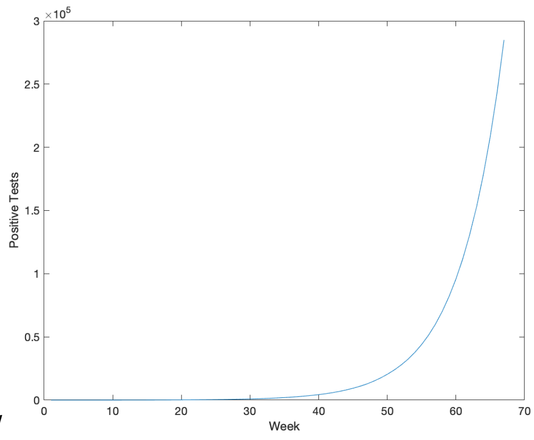


Figure 2.1

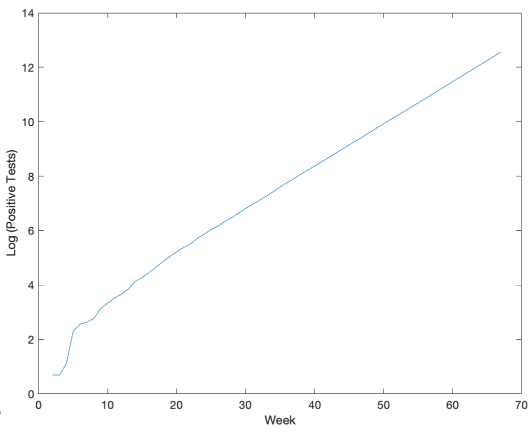


Figure 2.2

In *Figure 2.1* we see a plot of the number of people testing positive vs. time (week). It can be visually observed that the growth rate is exponential, as suggested in the question brief. In *Figure 2.2* we see a plot of the logarithm of the amount of people testing positive vs time (week). We can deduce that there is a linear relationship between that of the logarithm of positive cases and time. It should be noted that there are inconsistencies in the dataset which result in outliers when visually presented. We can see the effect of these outliers prior to *week 10*. This will be addressed further in the linear regression model analysis. In order to estimate the growth parameter a, we must calculate the slope of the line in *Figure 2.2*. In order to do this, we used a snippet of code in Matlab. The code selected two random integers from the max number of rows, and calculated the slope using the formula (y2-y1)/(x2-x1). It took the absolute values, and it was run three times, and the average value was found. The results were as follows:

a Value - Test 1	a Value - Test 2	a Value - Test 3	a Value - Average
0.1562	0.2719	0.1550	**0.1944**

Estimated growth parameter *a*: **0.1944**

(b)

```
X = (1:size(log(dataset.data(:,2)))).'; %create array of x values corresponding to time (week)
Y = log(dataset.data(:,2)); %create array of y values corresponding to the logarithm of positive tests
m = 0; %set slope to zero y=mx+C
c = 0; %set y-intercept to zero y=mx+C
```

```
stepSize = 0.0001; %set the learning rate of the model
iterations = 10000; %set the number of iterations for the model
for regIndex = 1:iterations
   yValuePrediction = (m*X) + c ; %create an array of Y value predictions
   derivativeSlope = (-2/length(X)) * sum(X .* (Y - yValuePrediction));
   derivativeYIntercept = (-2/length(X)) * sum(Y - yValuePrediction);
   m = m - stepSize * derivativeSlope;
   c = c - stepSize * derivativeYIntercept;
end
plot(logs);
hold on;
plot(X, m*X+c);
legend('Data', 'Linear Regression Model');
```

Explanation: The first thing we need to do is create arrays with the corresponding X and Y value for positive tests and time. We then initialise the slope and y-intercept to zero, as well as set our step size and number of iterations for the model. For the cost function, we are taking the sum of the square of the difference between the actual positive tests value, and the predicted positive test value, and finding the mean value for the dataset. This is called the Mean Squared Error.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

Mean Squared Error

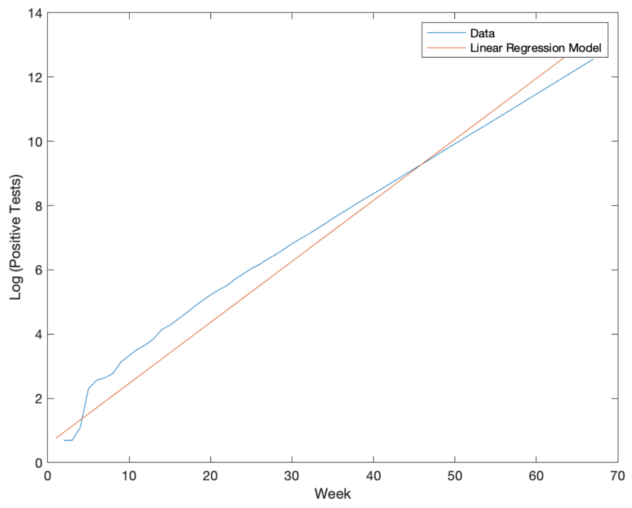
The predicted value of Y is the equation of a line, y=mx+c, as the relationship between the variables is linear. This is done in the first line in the for-loop for gradient descent. We then calculate the partial derivative with respect to the slope in order to produce values for both the derivative of the slope and the derivative of the y-intercept (c), which is then

$$\frac{1}{n}\sum_{i=1}^{n} -2x_i(y_i - (mx_i + b))$$

Partial Derivative of MSE

multiplied by the current slope after subtracting the learning rate. It should be noted that that learning rate is how much the slope will change for each iteration of the model. We then plot the original logarithmic function of the positive cases and the linear regression model in order to analyze the model further.

(c)



Estimated Growth Parameter (a): **0.1898**

Level Of Initial Infections (logx0): **0.5679**

Explanation: On the left, a graph with the original logarithmic function of positive tests has been plotted alongside the linear regression model. This was produced in-order to visually inspect the model, and assess any assumptions the model makes, and the errors that may be thrown. This is discussed in part (iii) of this section.

(i) If we look at the estimated growth parameter in part (a) of 0.1944, we cannot have a step size of 1 or 0.1 as we will most likely exceed the growth parameter and cause the model to behave erratically. If the step size is too small, the model will not be efficient. It should be noted that an optimal step size is that of one which achieves a balance between accuracy and efficiency. To confirm this, the model was run with a step size of 1 and 0.1, and returned a NaN value for *m*. It would then be the assumption that a step size of 0.01 would be appropriate, however as the step size in the model is applied on *c* (y-intercept) as well, it would result in a NaN output, as the step size would exceed the level of initial infection value. A step size of 0.001 would be appropriate, however to ensure further accuracy in the model, a step size of 0.0001 was selected. When performed with this step size for 1000 iterations, it was found that the model was heavily affected by outliers. So to ensure accuracy, the iterations were increased to 10000.

(ii) There are four main statistical assumptions associated with a linear registration model. These assumptions are: *linearity, homoscedasticity, independence,* and *normality*. Linearity is that the relationship between the logarithmic function of positive tests and time is linear. Homoscedasticity is that any error is the same across all values in the dataset, that is every positive COVID test has the same error value. Independence is that all positive COVID tests are independent of each other. Normality is the assumption that the data used in the model is normally distributed, which was discussed in a previous question.

(iii) In terms of Linearity, the data mostly agrees with the assumption that the relationship is linear. However, it should be noted that there are outliers present in the data, and as we are using the squared method of calculating the cost function, the effect of outliers in the data is magnified. In order to minimise this effect, the absolute method of calculating the cost function could be used - which effectively ignores the presence of outliers. We initially ran the model with 1000 iterations, although this was increased to 10000 iterations to minimise the effect of outliers and increase the accuracy of the model. We can visually observe the effect of this in the plot of the previous question, as well as numerically observe an increased slope caused by outliers. However, as the majority of the data is linear, we can predict that this assumption is satisfied. In terms of homoscedasticity, we can assume this assumption is satisfied as we were provided with a false positive figure of 0.01 that applied to the dataset in its entirety. Although, more information would be appreciated to confidently state the assumption is satisfied. We assume the data is independent as COVID tests must follow a strict procedure, although similar to homoscedasticity, we are lacking further information. In terms of the distribution of the data, we assume the data is normally distributed - however it is likely that this assumption is violated as due to the nature of COVID tests, there may be increased testing in areas such as nursing homes, which would have a majority of older persons with increased health conditions. Scenarios like this would cause the data to be skewed, and thus we assume that the condition of normality may be violated.

(d) When we bootstrap a linear regression model, we can understand how the model parameters vary. To bootstrap our model, we follow the concept of extracting several random samples from our dataset, and running the model in that closed sample. The regression model using gradient descent is performed, in this case with a sample size of 20 data points for 5 samples. Once this is complete the array of sample means and intercepts can be used to calculate the variance of the results and thus derive a confidence interval using the central limit theorem method which is 2 times the population standard deviation, and the Chebyshev method which is the population standard deviation divided by sqrt(0.05), which produces bounds with 95% confidence.

(e)

```
numberOfSamples = 5; %set number of samples to take
sampleSize = 20; %set the size of samples
sampleMeans = zeros(1,numberOfSamples); %initialise means array
sampleIntercepts = zeros(1,numberOfSamples); %initialise intercepts array
for bootIndex = 1:numberOfSamples
  sampleX = sort((randperm(length(logs),sampleSize))).'; %randomly select sample and sort
  sampleY = logs(sampleX); %find corresponding log positive test values
  sampleM = 0; %set slope to zero y=mx+C
  sampleC = 0; %set y-intercept to zero y=mx+C
  for regIndex = 1:iterations %perform gradient descent
     yValuePrediction = (sampleM*sampleX) + sampleC ; %create an array of Y value predictions
     derivativeSampleM = (-2/length(sampleX)) * sum(sampleX .* (sampleY - yValuePrediction)); %find derivative of sum of actual vs
predicted
     derivativeSampleC = (-2/length(sampleX)) * sum(sampleY - yValuePrediction); %find derivative of sum of actual vs predicted
     sampleM = sampleM - (stepSize * derivativeSampleM); %adjust m with step size and calculated derM
     sampleC = sampleC - (stepSize * derivativeSampleC); %adjust m with step size and calculated derC
  end
  sampleMeans(bootIndex) = sampleM; %save calculated sample slope
  sampleIntercepts(bootIndex) = sampleC; %save calculated sample intercept
end
populationMeanM = sum(sampleMeans)/length(sampleMeans);
populationStandardDevM = sqrt(var(sampleMeans));
m_clt_lowerBound = populationMeanM - 2*(populationStandardDevM);
m_clt_upperBound = populationMeanM + 2*(populationStandardDevM);
m_chebyshev_lowerBound = populationMeanM - populationStandardDevM/sqrt(0.05);
m_chebyshev_upperBound = populationMeanM + populationStandardDevM/sqrt(0.05);
populationMeanC = sum(sampleIntercepts)/length(sampleIntercepts);
populationStandardDevM = sqrt(var(sampleMeans));
c_clt_lowerBound = populationMeanC - 2*(populationStandardDevC);
c_clt_upperBound = populationMeanC + 2*(populationStandardDevC);
c_chebyshev_lowerBound = populationMeanC - populationStandardDevC/sqrt(0.05); c_chebyshev_upperBound = populationMeanC +
populationStandardDevC/sqrt(0.05);
```

```
Growth Parameter: CLT 95-Percent Confidence Interval is [0.187567, 0.194680]
Growth Parameter: Chebyshev 95-Percent Confidence Interval is [0.183170, 0.199076]

LogX0: CLT 95-Percent Confidence Interval is [0.272398, 0.702286]
LogX0: Chebyshev 95-Percent Confidence Interval is [0.006712, 0.967972]
```

For the code above, we used the same step size and iterations as the initial regression model in the previous question. The code then randomly selects a sample from the dataset and runs the regression model. This is repeated 5 times and the results are saved to an array. This array is then used to calculate the mean of the samples, as well as the variance and standard deviation. Finally we calculate the confidence intervals using the CLT method and Chebyshev method. As expected, the array of both the means and intercepts were similar to the initial regression model in the previous question. Both the mean and intercept values from the previous question, fall into the confidence interval. We can note that the Chebyshev confidence intervals have larger bounds, which is expected due to the nature of Chebyshev providing an actual bound, not an estimated bound which is what the CLT method provides. An interesting observation was that our initial rough estimate of the growth parameter *a* was **0.1944** in part (a). This was an estimate provided by manually calculating the slope. As much as this estimate falls into the bounds of the confidence interval, it is very close to exceeding the upper bound of both the CLT and Chebyshev interval.

(f)

```
valueBetweenInterval = 0.183170 + (0.199076-0.183170).*rand(100,1); %select range of 100 values between Chebyshev interval
valueArray = exp(valueBetweenInterval .* 10)*0.5679; %calculate xk = e^ak(x0), note take x0 = 0.5679
populationMeanK = sum(valueArray)/length(valueArray); %calculate mean
populationStandardDevK = sqrt(var(valueArray)); %calculate std
c_chebyshev_lowerBound = populationMeanK - populationStandardDevK/sqrt(0.05); %Chebyshev Lower Bound
c_chebyshev_upperBound = populationMeanK + populationStandardDevK/sqrt(0.05); %Chebyshev Upper Bound
```

Xk when k = 10: Chebyshev 95-Percent Confidence Interval is [3.149411, 4.570982].

First we select a range of 100 values that are within the Chebyshev bounds. We are using the Chebyshev bounds as they are actual bounds, not estimates. This is calculated by following the equation: lower bound + (upper bound - lower bound) * random value between 0-1. We then sub our array of values into the equation in the question and an array of Xk values are returned. Similar to the previous part, we calculate the mean and standard deviation, followed by the Chebyshev 95% confidence interval. If we analyse the logarithmic function of week 10 in our original dataset, we have a value of 3.3322. As we can see, this value falls within our interval. As much as this does not provide any further insight, it allows us to confirm that our interval is acceptable in comparison to the original dataset.

Appendix

```
dataset = importdata('/Users/ethanvictormonkhouse/final2021.php'); %import data

%%% Stochastic Simulation %%%

n = 100000; %run simulation n times
test_positive_p = max(dataset.data(:,2))/max(dataset.data(:,1)); %the probability of testing positive
test_positive_with_s_p = max(dataset.data(:,3))/max(dataset.data(:,2)); %the probability of having symptoms with a positive test

r=rand(1,n); %generate either a 0 or 1, n times
tested=(r<=(test_positive_p)); %label whether each value of r is positive or not
r=rand(1,sum(tested)); %for the amount of positive cases, generate a 0 or 1
test_positive=(r<(test_positive_with_s_p)); %label whether each value of r has symptoms or not

tp_sim_s_p = sum(test_positive)/n; %all positive with symptoms cases / population of simulation
tp_sim_ws_p = (length(test_positive) - sum(test_positive))/n; %all positive without symptoms cases / population of simulation

%%% Plot Dataset %%%

plot(dataset.data(:,2)) %plot positive tests vs time
xlabel('Week') %label axis
ylabel('Positive Tests') % label axis

logs = log(dataset.data(:,2)); %calculate log (positive tests) vs time
plot(logs); %plot logs
xlabel('Week') %label axis
ylabel('Log (Positive Tests)') %label axis

random1 = randi(67); %generate random value
random2 = randi(67); %generate random value
estimate_a = abs((logs(random2)-logs(random1))/(random2-random1)) %calculate slope (y2-y1)/(x2-x1)

%%% Linear Regression Model %%%

X = (1:size(log(dataset.data(:,2)))).'; %create array of x values corresponding to time (week)
Y = log(dataset.data(:,2)); %create array of y values corresponding to the logarithm of positive tests
Y(1) = 0; %logarithm of zero is infinite, we must exclude from our model

m = 0; %set slope to zero y=mx+C
c = 0; %set y-intercept to zero y=mx+C

stepSize = 0.0001; %set the learning rate of the model
iterations = 10000; %set the number of iterations for the model

for regIndex = 1:iterations
    yValuePrediction = (m*X) + c ; %create an array of Y value predictions
    derivativeSlope = (-2/length(X)) * sum(X .* (Y - yValuePrediction)); %find derivative of sum of actual vs predicted
    derivativeYIntercept = (-2/length(X)) * sum(Y - yValuePrediction); %find derivative of sum of actual vs predicted
    m = m - stepSize * derivativeSlope; %adjust m with step size and calculated derM
    c = c - stepSize * derivativeYIntercept; %adjust m with step size and calculated derC
end

plot(logs);
hold on;
plot(X, m*X+c);
xlabel('Week') %label axis
ylabel('Log (Positive Tests)') %label axis
legend('Data', 'Linear Regression Model');
hold off;

%%% bootstrapping %%%

numberOfSamples = 5; %set number of samples to take
sampleSize = 20; %set the size of samples
```

```
sampleMeans = zeros(1,numberOfSamples); %initialise means array
sampleIntercepts = zeros(1,numberOfSamples); %initialise intercepts array

for bootIndex = 1:numberOfSamples
   sampleX = sort((randperm(length(logs),sampleSize))).'; %randomly select sample and sort
   sampleY = logs(sampleX); %find corresponding log positive test values
   sampleM = 0; %set slope to zero y=mx+C
   sampleC = 0; %set y-intercept to zero y=mx+C
   for regIndex = 1:iterations %perform gradient descent
      yValuePrediction = (sampleM*sampleX) + sampleC ; %create an array of Y value predictions
      derivativeSampleM = (-2/length(sampleX)) * sum(sampleX .* (sampleY - yValuePrediction)); %find derivative of sum of actual vs predicted
      derivativeSampleC = (-2/length(sampleX)) * sum(sampleY - yValuePrediction); %find derivative of sum of actual vs predicted
      sampleM = sampleM - (stepSize * derivativeSampleM); %adjust m with step size and calculated derM
      sampleC = sampleC - (stepSize * derivativeSampleC); %adjust m with step size and calculated derC
   end
   sampleMeans(bootIndex) = sampleM; %save calculated sample slope
   sampleIntercepts(bootIndex) = sampleC; %save calculated sample intercept
end

%%% confidence interval %%%

populationMeanM = sum(sampleMeans)/length(sampleMeans);
populationStandardDevM = sqrt(var(sampleMeans));

m_clt_lowerBound = populationMeanM - 2*(populationStandardDevM); %CLT Lower Bound
m_clt_upperBound = populationMeanM + 2*(populationStandardDevM); %CLT Upper Bound
fprintf("CLT 95-Percent Confidence Interval is [%f, %f]. \n", m_clt_lowerBound,m_clt_upperBound); %print

m_chebyshev_lowerBound = populationMeanM - populationStandardDevM/sqrt(0.05); %Chebyshev Lower Bound
m_chebyshev_upperBound = populationMeanM + populationStandardDevM/sqrt(0.05); %Chebyshev Upper Bound
fprintf("Chebyshev 95-Percent Confidence Interval is [%f, %f]. \n", m_chebyshev_lowerBound,m_chebyshev_upperBound); %print

populationMeanC = sum(sampleIntercepts)/length(sampleIntercepts);
populationStandardDevC = sqrt(var(sampleIntercepts));

c_clt_lowerBound = populationMeanC - 2*(populationStandardDevC); %CLT Lower Bound
c_clt_upperBound = populationMeanC + 2*(populationStandardDevC); %CLT Upper Bound
fprintf("CLT 95-Percent Confidence Interval is [%f, %f]. \n", c_clt_lowerBound,c_clt_upperBound); %print

c_chebyshev_lowerBound = populationMeanC - populationStandardDevC/sqrt(0.05); %Chebyshev Lower Bound
c_chebyshev_upperBound = populationMeanC + populationStandardDevC/sqrt(0.05); %Chebyshev Upper Bound
fprintf("Chebyshev 95-Percent Confidence Interval is [%f, %f]. \n", c_chebyshev_lowerBound,c_chebyshev_upperBound); %print

%%% range of values %%%

valueBetweenInterval = 0.183170 + (0.199076-0.183170).*rand(100,1); %select range of 100 values between Chebyshev interval
valueArray = exp(valueBetweenInterval .* 10)*0.5679; %calculate xk = e^ak(x0), note take x0 = 0.5679

populationMeanK = sum(valueArray)/length(valueArray); %calculate mean
populationStandardDevK = sqrt(var(valueArray)); %calculate std

c_chebyshev_lowerBound = populationMeanK - populationStandardDevK/sqrt(0.05); %Chebyshev Lower Bound
c_chebyshev_upperBound = populationMeanK + populationStandardDevK/sqrt(0.05); %Chebyshev Upper Bound
fprintf("Chebyshev 95-Percent Confidence Interval is [%f, %f]. \n", c_chebyshev_lowerBound,c_chebyshev_upperBound); %print
```