

Study of a Manufacturing Facility

Project Deliverable II (Starts on Page 12)

SYSC 4005

Group: #13

Anthony Maevski-Popov 100981314

Wasam Al-bayti 101041465

Chang Qiu 100997188

1.0 - Problem Formulation

The purpose of our simulation study is to determine if the performance of the proposed manufacturing facility's model is optimal or not. If the model is not optimal, it is essential to determine how to change the current design and maximize efficiency. A workstations buffer(queue) size is too small, the current policies being inefficient, and inspectors' current priorities are all potential problems reducing the efficiency of the design.

2.0 - Setting of Objectives and Overall Project Plan

2.1 Setting Objectives

The objectives for this simulation project are to observe the performances of the manufacturing facilities, evaluate the work stations' service times and improve the policies. According to the formulation and objectives, simulation is the appropriate methodology. Simulation is an imitation of the real-life system. The output of the simulation model corresponds to the output of the real-world system. By simulating the manufacturing facility, we can explore and assess the ongoing system without interrupting it to set new policies and new rules.

2.2 Proposed Plan Alternative system design:

A design could be implemented to increase the processing times for the three workstations where an additional inspector is added to the system, which will focus on component 3 only. This will leave inspector 2 to focus solely on component 2. By adding a third inspector the randomness of picking a component to work on by inspector 2 is removed.

2.3 Proposed plan for the study

The simulation study will be conducted through a series of 4 milestones. The project plan is as follows:

2.3.1 - Milestone One

The first step before conducting our simulation study is to determine the goal or purpose of the study. This was achieved in “**1.0 - Problem Formulation**” above. Next, we will determine the nature of the system to understand how it currently functions. Once we have an understanding of how the real-world system operates, we will translate that system model into a program. Once we have a working program of the system model, we can begin to perform data collection and input modeling.

2.3.2 - Milestone Two

As we were given historical data sets for each of the inspector's service times and the workstations's service times, we will perform statistical analysis on each of the data sets and determine the distribution that the data sets follows. Then we need to recognize the data distribution, whether it is a uniform distribution, an exponential increase, etc. From these distributions we will perform the Q-Q plots and chi-square goodness of fit test to determine the best distribution that fits the data set. With this knowledge we can then change our program to generate data that follows the best model identified.

2.3.3 - Milestone Three

The next step is the verification of the model. It can be achieved by running simulations to inspect whether the computer program and the input parameters of the model are performed properly. If the model is not verified correctly, the group will take a step back and make sure that the code is translated correctly from the model. Validation will be achieved by calibrating the model. The model will be iteratively compared to the actual system behaviour, and updated after each iterative process until it behaves efficiently and adequately. Furthermore, production runs and analyses will be conducted to measure the performance of the simulated design.

2.3.4 - Milestone Four

In the final milestone, we will introduce an alternate operating policy. We will create this policy to improve the overall performance of the proposed system and conduct further evaluations to determine the impact of these improvements on the system. We will then write up a project conclusion. In the project conclusion, we will talk about the simulation study process, the major discoveries made, our improvements to the model, and the efficiency of the model system. The project will be documented by

program and progress. It is essential to document the project in case the program is accessed by a different analyst who wishes to improve the system further.

3.0 - Model Conceptualization

Currently, the system model, presented to us from the system description, is a "Manufacturing Factory" containing two inspectors and three workstations. The inspectors have an infinite supply of the components. They are responsible for forwarding the components to the workstation's buffers (queues) immediately if the buffers are not full.

Inspector 1 is responsible for creating the "C1" component. Inspector 2 is responsible for creating components "C2" and "C3" respectively. Inspectors have an unlimited supply of components and can work continuously. Each "P1" product requires 1 C1 component in Workstation 1. Workstation 2 requires one component of C1 and C2 to produce a "P2". Workstation 3 requires C1 and C3 to produce a product "P3". Each workstation has a buffer (queue) of size 2 for each component required to make a product. Each workstation may only produce a product when it has at least one of each component for the product in the buffer (queue). If multiple workstation's buffers (queue's) for component C1 is not full, the inspector 1 who creates C1 will prioritize Workstations buffers with the least components. In the event of a tie, Inspector 1 will prioritize in Workstations in the following order respectively, (W1,W2,W3).

The service times for an inspector represent the time spent inspecting a component before sending it to one of the workstation's buffers, based on a policy that each inspector follows. As previously mentioned, currently inspector one follows a routing policy and then a priority policy, if a tie occurs. If all workstation buffers(queue) for component 1 are full, inspector 1 will WAIT (blocked) until a buffer has space before inspecting the next component for a given "service time." Inspector 2 currently will pick at random what component to inspect from component (C2,C3), and then after inspection, send that component to the buffer associated with that component. In the event that the buffer for the component the inspector is trying to forward to is full, the inspector will be in a WAITING state (blocked) until the buffer has room before inspecting a new random component for a given serviceTime. As mentioned before, workstations will be in a WAITING (blocked) state until it has at least one of each component required to craft a Product in their component Buffer(s). Once this condition is met, workstations will begin producing a product for a length of time. This length is the

service time. Once the service time passes, the product is deemed “assembled”, and one component from each buffer is removed.

The actors of the system are the inspectors who are sending components to the buffer (queue). The server are the workstations receiving the components required to produce their products.

4.0 - Model Translation

In this section, we will explain our choice of language, how we implemented our model, with a description.

4.1 - Our Choice of Language

As a group, we have decided to potentially use **two** languages: “Java” and “MatLab” to as our simulation environments for the “Manufacturing Factory Model.” Java is a high-level programming language that has many libraries as well as many external support libraries available for us, making the implementation of this model and further implementation of policies or data generation from input modeling easy.

Furthermore, MatLab provides data types and pre-processing capabilities to process engineering and scientific data through data visualization and machine learning. It can expand the analysis to big data without further code change. Therefore, MatLab is useful for our project to examine data sets, compare data sets, and generate Q-Q plots and chi-squares. Also, MatLab’s built-in function Simulink is also useful in this project. Simulink is a MatLab based programming environment for modeling, simulation and analyzing. We will use this program to develop simulation models, run simulation models, and compare results in MatLab and Java.

4.2 Implementation of the model

The model is implemented through the use of classes in java. The model uses the data from the historicalData files in order to run the simulation. The Following are the java classes, used to simulate the model and the responsibilities of each class.

The detailed implementation of the model will be presented in the following sections with an illustration of how the implementation of the model follows the model.

4.2.1 Components.java

Currently holds each type of the current components in the mode, currently “C1,C2,C3”. This referenced by the inspector’s to know what component they are inspecting and the WorkStations to know what components they need.

4.2.2 Products.java

The products class holds each type of the current products in the model. Currently, it contains “P1,P2,P3” respectively. The WorkStations class refers to the product class to know what product they produce.

4.2.3 FactoryObjects.java

As java is an object-oriented language, we can treat many things as objects. Objects have states and behaviours. For instance, Inspectors and Workstations are both objects.

The purpose of the FactoryObject Class is to examine the various behaviours and attributes shared in all FactoryObjects in the “manufacturing factory model” given.

For all object types, currently Inspectors and WorkStations, FactoryObject.java has the following shared characteristics: a name, a state, the duration in that state, the serviceTime remaining and the number of tasks completed.

The methods in FactoryObject.java mostly contain getters and setters for most of the characteristics listed above.

- ❑ Getting the service time remaining, the number of tasks completed
(example: inspections completed or products produced)
- ❑ getting the state of a object
 - ❑ the inspector actively working/inspecting a component
 - ❑ the inspector in WAITING state , when the buffer is full

All of these attributes and methods get inherited by the “Inspectors.java and WorkStations.java” classes.

4.2.4 ObjectStates.java

Holds each possible type of state FactoryObjects can have. Currently, they are of "STARTING_UP, WAITING, ACTIVE, IDLE". Currently used by both the Inspectors and Workstations for recording what state they are in.

4.2.5 SimulateFactoryModel.java

This class is where we setup the objects needed for the simulation and simulate its eventual conclusion.

Currently, this class is responsible for creating the 5 Factory Objects required for the simulation, currently I1, I2, WS1, WS2, WS3, for the 2 inspectors and 3 workstations, respectively.

During the setup process, we tell the workstations:

- What components they will be working with for them to make a product.
- The buffer or queue size for the number of components that can be held by each component type that a workstation will need.
- The service times (currently processed sequentially from the historical data files, telling the workstations how long they will spend (serviceTime) in assembling a product, once it has the required components.

We also create the 2 inspectors and tell them:

- The workstations they have access too
- The priority level for when they are done inspecting a component and want to place it onto a workStations buffer (queue)
- The ServiceTimes, once again read from the historical Data files sequentially, on how long an inspector is to imitate inspecting a component for.

We also have a method to read data from the historical Data files, line by line.

Lastly, we have our "Main " where we run our simulation until all Factory Objects are either in an IDLE or WAITING state. For example, (in the simulation run, I will discuss later), Inspector#1 was found to be IDLE because it ran out of component #1's. The remaining inspector was waiting because the buffer for workstation 2 or 3 or both, were full. Moreover, the three workstations were also in the waiting state, as they were waiting for component #1 that would never come because inspector#1 ran out of component #1 to inspect.

4.2.6 WorkStations.java

This class is where we run our workstations. After “SimulateFactoryModel.java” starts running, three workstation objects are created. When a workstation object is created, the constructor records product it can produce, the components buffers it requires as well as setting up the buffer queue, the buffer size limit, as well as the service times required to produce a product (service times in sequential order).

Once active, workstations will continue to monitor their state. Workstations will be in the WAITING state until it determines that all buffers assigned to a workshop have at least 1 component (since we need one of each component to make a product). When this is true, it will change to the active state where it will be in the process of producing a product for a set “serviceTime”, continuously updating the service time duration spent producing a product. Once it has waited for the serviceTime, it will deque all buffers by one component and then go back to the waiting state if all buffers for that workstation don't have at least one component.

This process is done through various methods:

- “updateComponentBuffers” used to update the buffers when a product is made.
- “tryToMakeProduct” used to check if a component exists within each buffer and its in the correct STATE.
- “isBufferFull” check to see if the workstation's buffer is at the buffer size limit (in our case the limit is 2).
- “addComponentsToComponentBuffer()” increments a specific component buffer by one.

4.2.7 Inspectors.java

This file is where the inspector objects are created. After “SimulateFactoryModel.java ” starts running, two inspectors objects are created.

When an inspector object is created, the constructor records:

- The components it can produce and to what workstations it can send that component too
- The number of components it has (in this case, there are 300 of each component since there are 300 “service times” in the historical Data files for each component.
- The workstations priority policy (used for inspector#1).





- The service times required for an inspector to inspect a component of the type they are working with(service times in sequential order).

Once an inspector is in the active state, they will take a component for inspection. Currently, for any inspector who has multiple components (currently only inspector#2), the component will be chosen at random using java's ".nextInt" method to determine a component from the ones assigned to the inspector. The inspector will then "inspect" (service time) the component and then try to send that component to a workstation depending on its policies. In the event that all component buffers for the component the inspector finished inspecting is full, the inspectors shift to a WAITING state, where it will continue to try to put a component on a buffer, until a buffer has room. At this point the inspector will have sent that component and then acquire the next component and its service time and then repeat the process until an inspector is either out of components to inspect, at which point it will enter the IDLE state or become stuck in the WAITING state and can't send a component, as mentioned in the last paragraph of "4.2.5".

- This process is done through various methods like "getComponentsForInspection()" which determines the next component an inspector needs to inspect.
- In the case of inspector#2 it used a random number generator to determine the next component to inspect, and then get the next service time of the component it chose.
- "tryToPutComponentToWorkStationsBuffer" check to see if the workstation buffer has room for the inspector to add a component to that buffer (queue).
- "getNextWorkStation()" which determines the workstation to send a component too based on inspectors policy.

4.3: Source Code Files and Basic Discussion of a Simulation Run

The source code for each module can be found in the “Program Files” folder.

	classes	2020-02-06 10:42 ...	File folder	
	historicalData	2020-02-06 5:20 AM	File folder	
	Sample Simulation Results.png	2020-02-06 11:32 ...	PNG image	52 KB
	SimulateFactoryModel.java	2020-02-07 12:31 ...	JAVA File	8 KB

- The class where users run the simulation process is “SimulateFactoryModel.java”
- All other classes are located in the classes folder.
- The historical data used in the simulation is located in the historicalData folder.
- A picture of the results of a a simulation trial using that data can be found in the “Sample Sinumation Results.png” file as well as the image below:

```

The Object: [Inspector#1]
Preformed this many component inspections: [300]
Were RESTRICTED from working due to the buffers(queue) being full for this long: [0.000] minutes
The Inspectors ACTIVE for a total of: [3107.381] minutes
Were Idle for: [%: 0.000] of the total time

The Object: [Inspector#2]
Preformed this many component inspections: [79]
Were RESTRICTED from working due to the buffers(queue) being full for this long: [1593.359] minutes
The Inspectors ACTIVE for a total of: [3121.537] minutes
Were Idle for: [%: 51.044] of the total time

The Object: [WorkStation#1]
Produced a total of: [222] products.
With an avarage production of: [4.27] Products Made/hour

The Object: [WorkStation#2]
Produced a total of: [46] products.
With an avarage production of: [0.88] Products Made/hour

The Object: [WorkStation#3]
Produced a total of: [31] products.
With an avarage production of: [0.60] Products Made/hour

```

After Running our simulation it was found that:

NOTE: the quantities of interest (Idle time and product output / unit of time) can be seen from the image above as well as outlined in the blue text below.

1. Inspector#1 was always active. As soon as it finished inspecting a component in its service time, there was always a buffer-free to take the next component.
2. The total number of components produced by the 3 workstations was $222+46+31$. The result equals to 299, yet Inspector #1 inspected and sent out 300 component 1's, so why is it 299?

Because a total of 299 products have been produced we can conclude that the last component #1 must exist either in workstation #2 or Work Station #3. This is because workstation #1 only requires a single component #1 to create a product #1.

What then could have happened is:

Workstation#2's buffer for component #2 was full and component#1 was empty. Inspector #2 randomly selected a component#2 to inspect but was stuck in an infinite WAITING STATE (blocked), since workstation#2 could not consume a component#2 to create a product#2, since its component#1 buffer was empty. This would also mean that at the time Workstation#3 had the last component#1 in its component#1 buffer but could not produce a product#3 since its component#3 buffer was empty.

3. On when simulating historical data Inspector#2 was idle for about 51%, spending most of its time most likely waiting for a full Buffer they to put a component on open up some space.
4. Workstations 1,2,and 3 produced 4.77, 0.88, 0.60 products / hour respectively.
5. Inspector#1 inspected 300 (all of his component#1's), while inspector #2 only inspected 79 of the 600 (300+300) of component 2 and component#3.

Might be useful later: (keep this here but remove it from Iteration#1 submission)

The following changes may improve the performance of the model :

- Invert inspector#1 priority policy.
- If Inspector#2 runs out of components #2 or component#3, notify inspector#1 not to send any more components to that workstation. Maybe add a workshop policy to return any excess component#1's back to the inspector if for example: workshop 2 used all of its components.
- Decrease workstation#1's buffer size from 2 to 1.
- Increase workstation 2 and 3 component #2 and #3 buffer size to 4 or 5 to reduce the probability of inspector #2 WAITING to put a component in a buffer that's full thus being unable to start inspecting the next component (reduce inspector#2 idle time)
- Make Inspector#2 alternate component's it inspects instead of randomly selecting them.

Deliverable II

5.0 Input Modeling

Data has been collected for different components and workstations in the working environment such as the inspection time of the three components, C1,C2 and C3. The data for assembly time of the workstations were also plotted as a histogram, W1,W2 and W3. The number of bins that are used in the histograms below is 18 (sqrt(300)). The Q-Q plot for the 6 different aspects of the environment were plotted using MATLAB and also a chi-square goodness-of fit test for each distribution was performed using MATLAB.

5.1 Inspection time for Components

5.1.1 inspection time for C1

The given data for the component was plotted in matlab with the simple code shown below:

```
histogram(ServiceTimesComponentA);
```

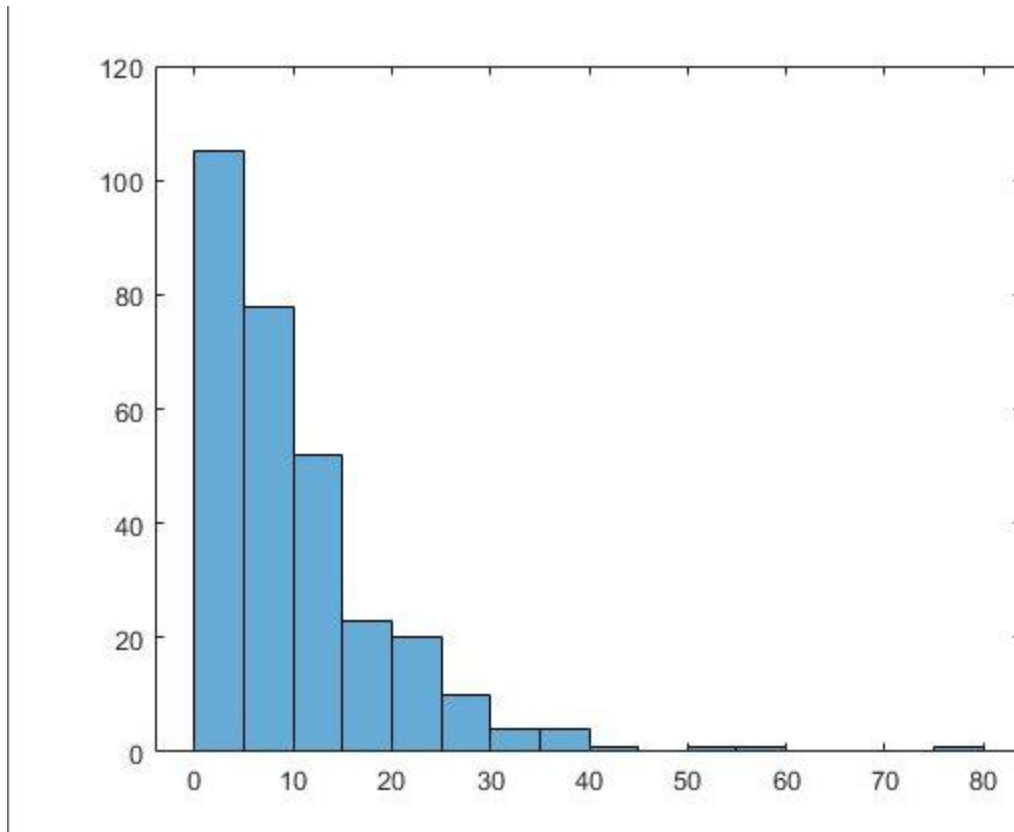


Figure 1: Histogram for Component 1

The above histogram could represent various types of distributions such as exponential, Weibull or Gamma distribution. Therefore, a Q-Q plot is needed to determine which type of distribution best fits the data points. Various distributions were tested which are shown below:

Code used for Q-Q plot:

```
expDist = makedist('Empirical');  
qqplot(ServiceTimesComponentA,expDist);
```

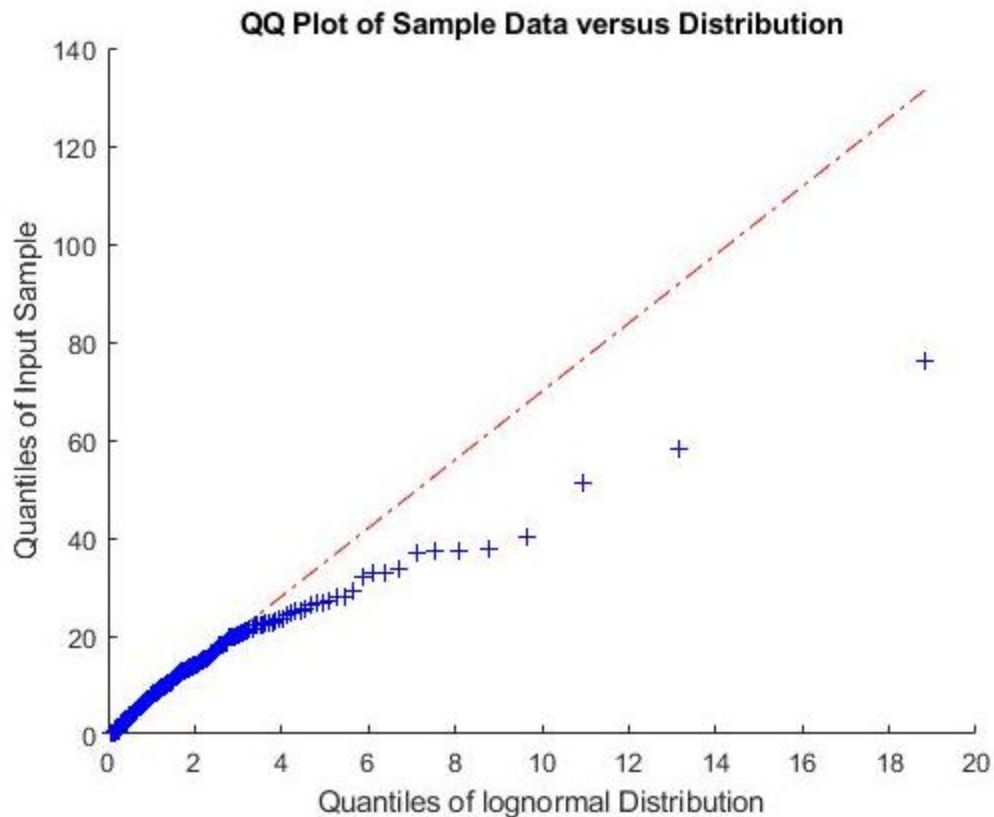


Figure 2: Q-Q plot for LogNormal Distribution

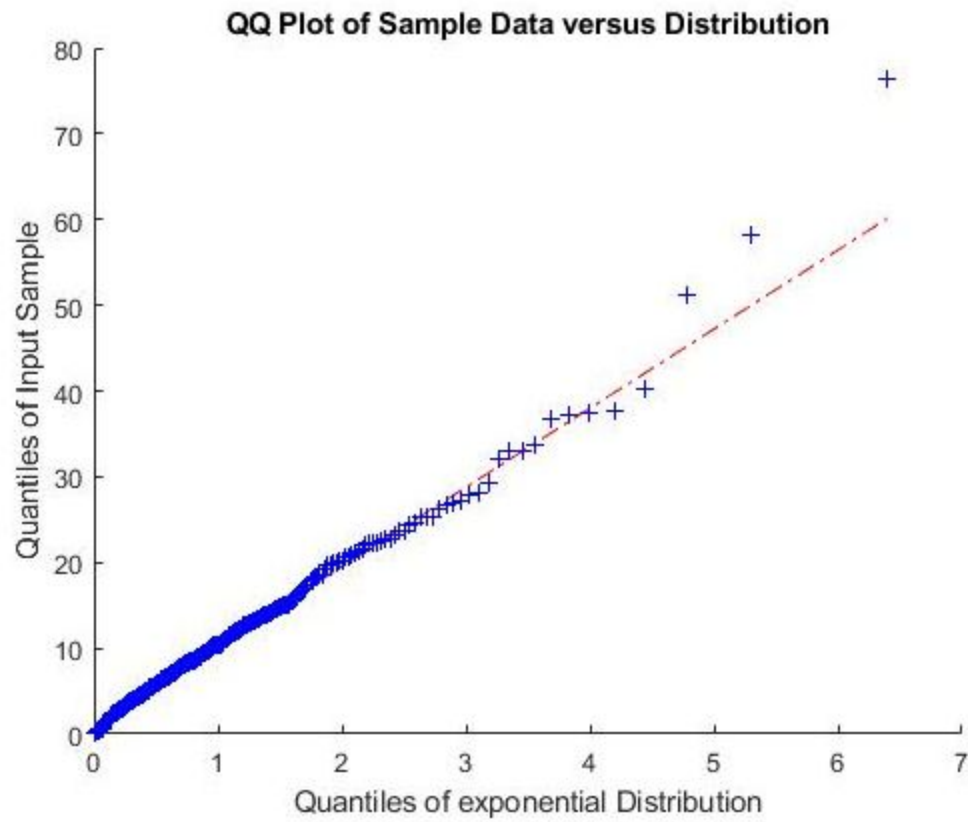


Figure 3: Q-Q plot for Exponential Distribution

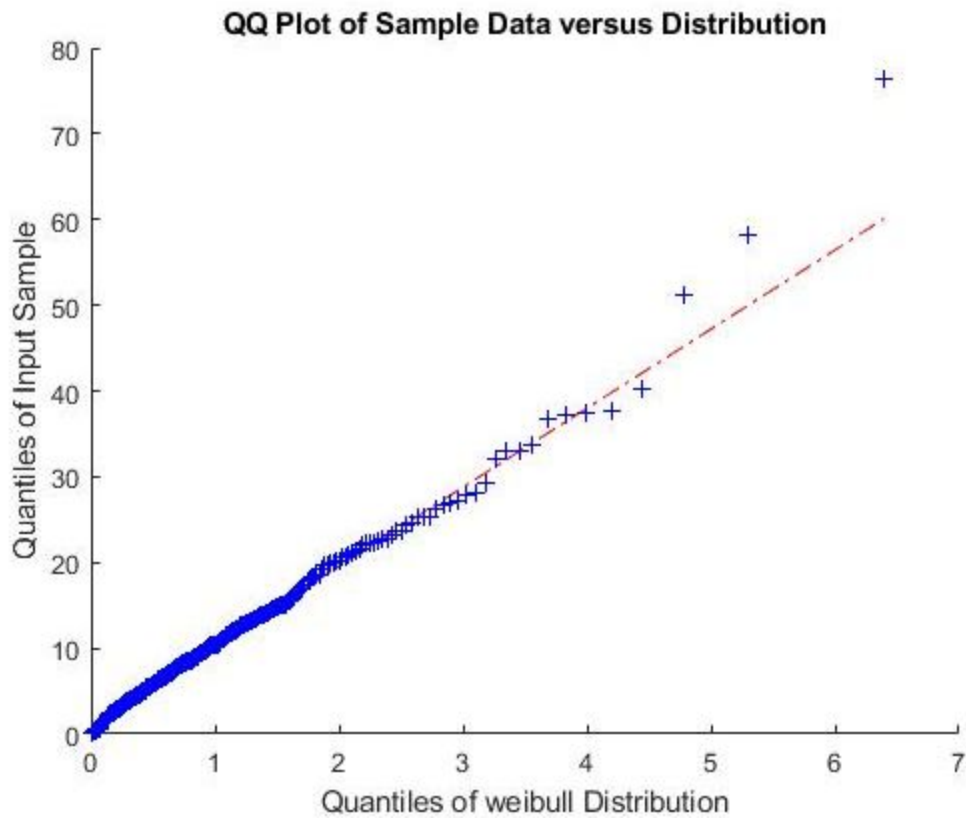


Figure 4: Q-Q plot for Weibull Distribution

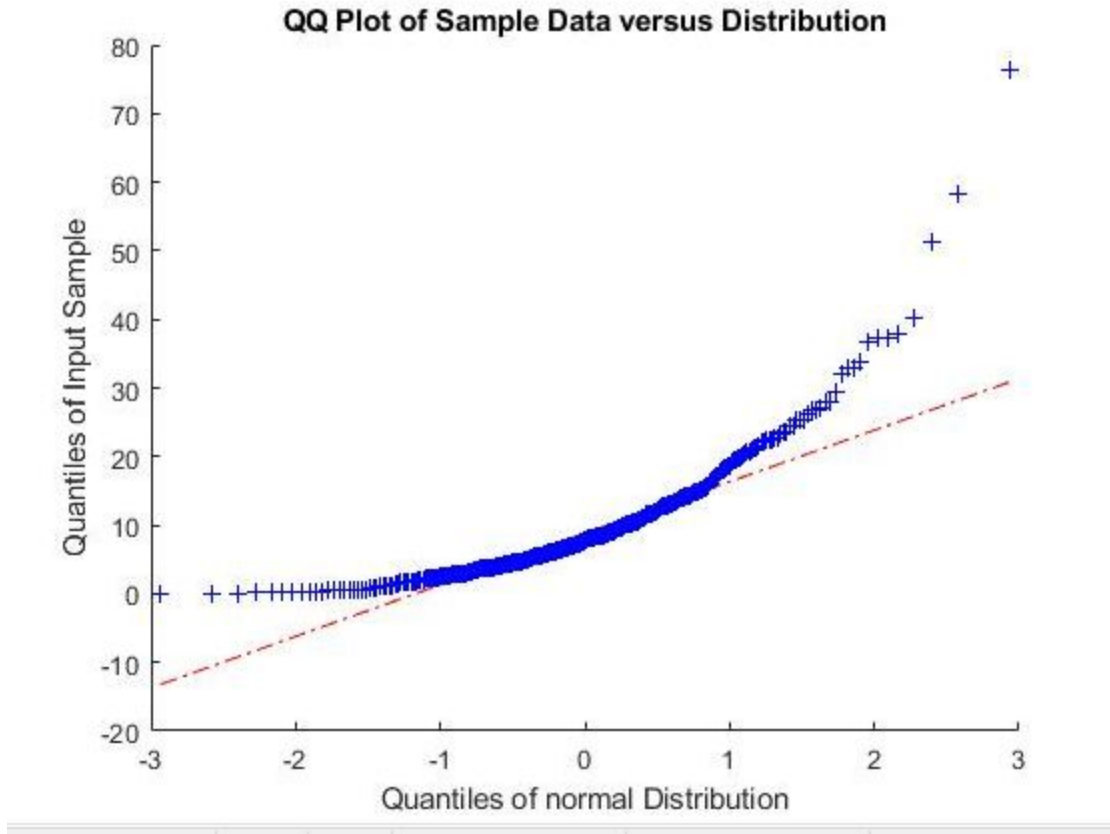


Figure 5: Q-Q plot for Normal Distribution

It can be seen above that the exponential distribution and the Weibull distribution both provide a linearity in the Q-Q plot whilst the lognormal and normal are non-linear.

A chi-squared test must now be performed to help with the choosing of the correct distribution. The code below is used to determine if the test passes or not:

```
pd = fitdist(ServiceTimesComponentA, 'Weibull'); %% for chi2gof, distribution can be changed  
chiTest = chi2gof(ServiceTimesComponentB, 'CDF', pd)
```

Whilst running this code, the value for pd is 1 for log and lognormal which means reject the hypothesis for that specific distributions tested. Pd was 0 for exponential and Weibull which means it passes the hypothesis, so both exponential and Weibull could be used to represent the inspection times for component 1, C1.

5.1.2 inspection time for C2

The code for plotting the histograms and Q-Q plots were the same as the C1. The histogram is shown below for component 2.

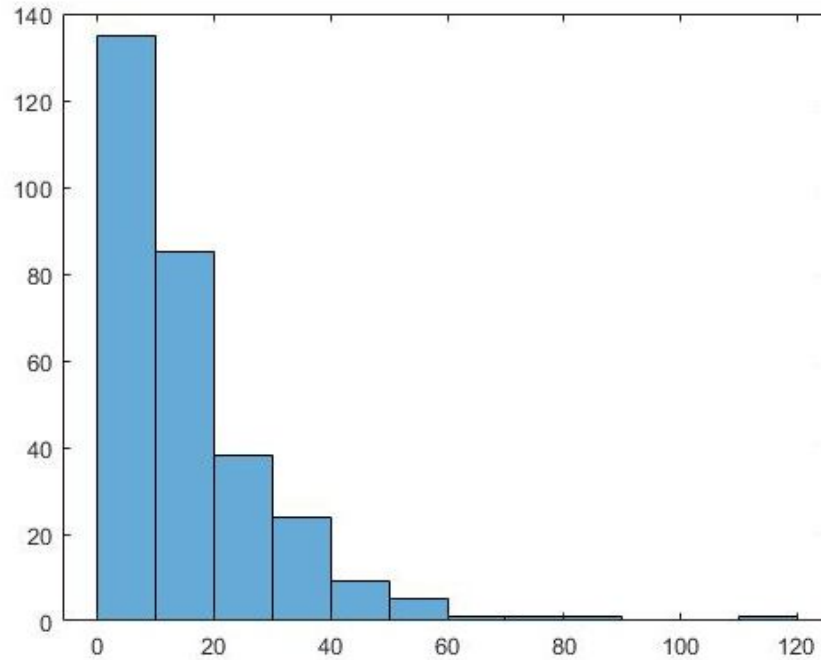


Figure 6: Histogram for Component 2

The histogram above is again similar to the first component.

The Q-Q plots below are shown for various distributions to see which distribution perfectly fits the data points linearly.

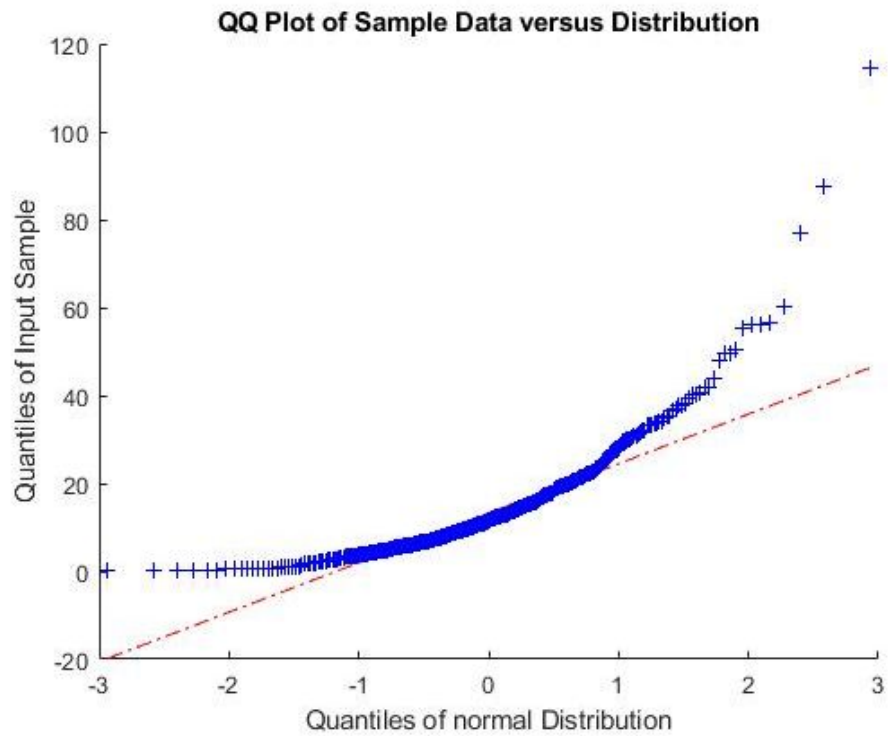


Figure 7: Q-Q plot of Normal Distribution

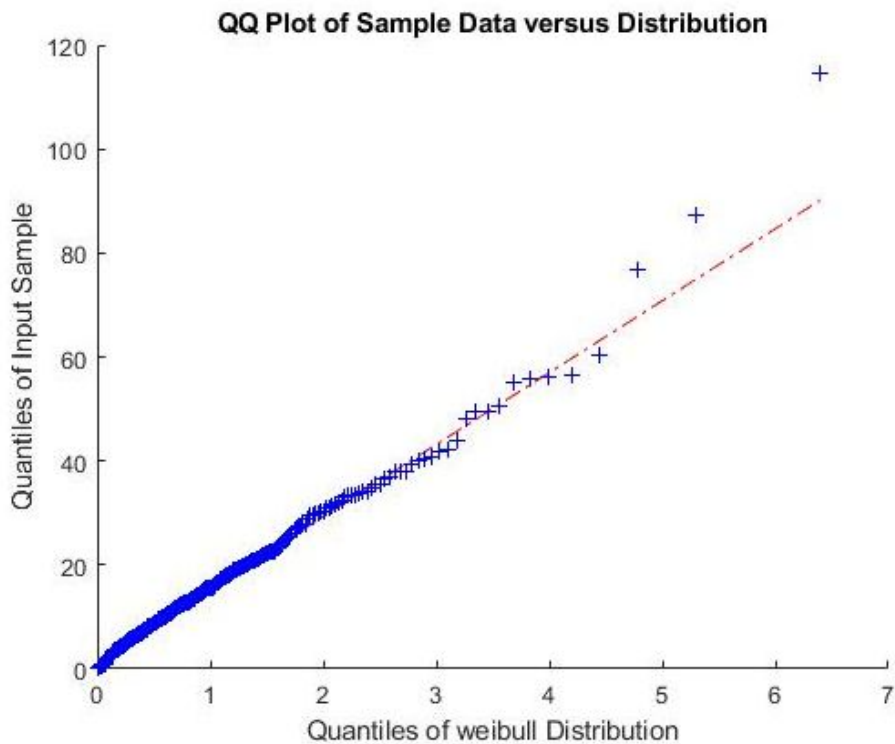


Figure 8: Q-Q plot of Weibull Distribution

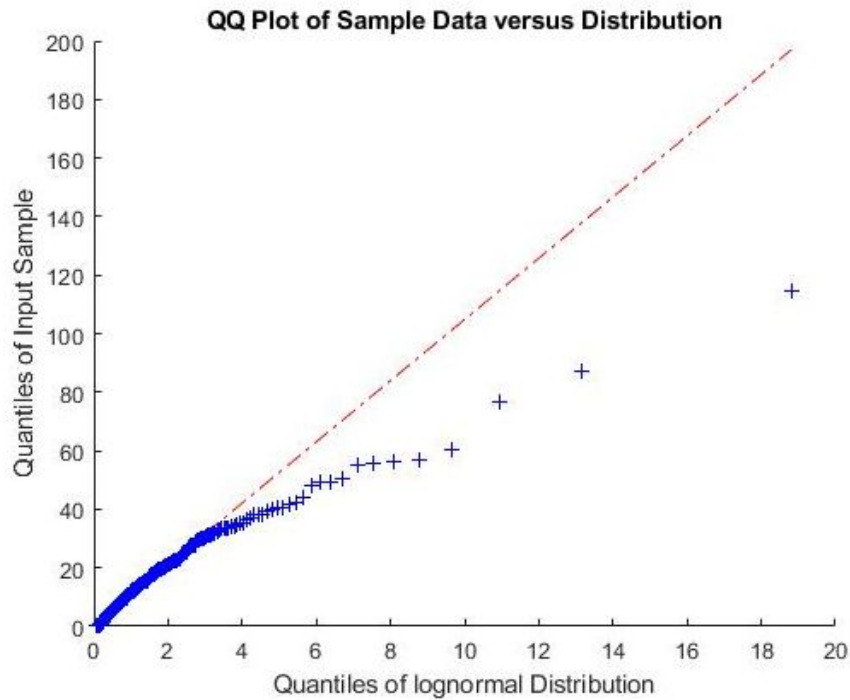


Figure 9 : Q-Q plot of Lognormal Distribution

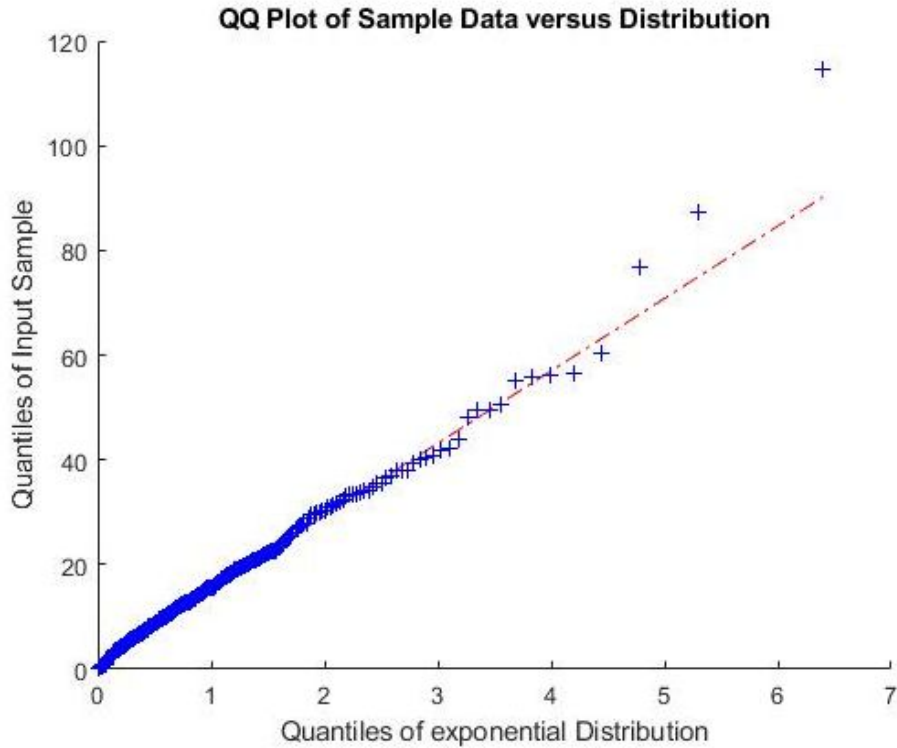


Figure 10: Q-Q plot of Exponential Distribution

It can be seen above that the exponential distribution and the Weibull distribution both provide a linearity in the Q-Q plot whilst the lognormal and normal are non-linear.

A chi-squared test must now be performed to help with the choosing of the correct distribution. The code below is used to determine if the test passes or not:

```
pd = fitdist(ServiceTimesComponentB, 'Weibull'); %% for chi2gof, distribution can be changed  
chiTest = chi2gof(ServiceTimesComponentB, 'CDF', pd)
```

Whilst running this code, the value for pd is 1 for log and lognormal which means reject the hypothesis for that specific distributions tested. Pd was 0 for exponential and Weibull which means it passes the hypothesis, so both exponential and Weibull could be used to represent the inspection times for component 2, C2.

5.1.3 inspection time for C3

The code for plotting the histograms and Q-Q plots were the same as the C1. The histogram is shown below for component 2.

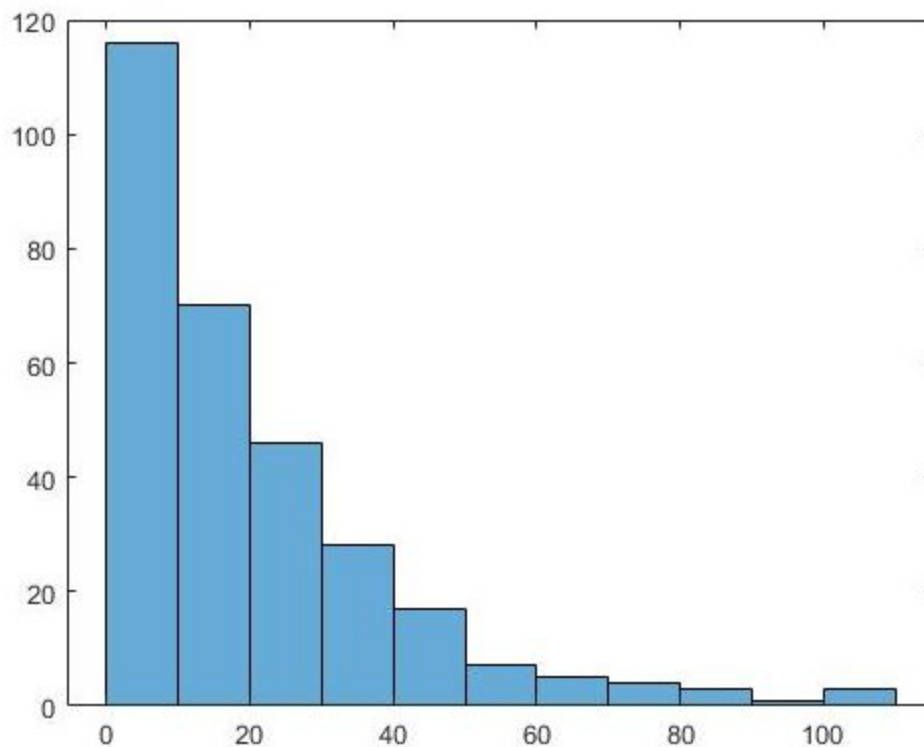


Figure 11: Histogram for Component 3

The histogram once again has a similar distribution to the other 2 components.
The Q-Q plots below are shown for various distributions to see which distribution perfectly fits the data points linearly.

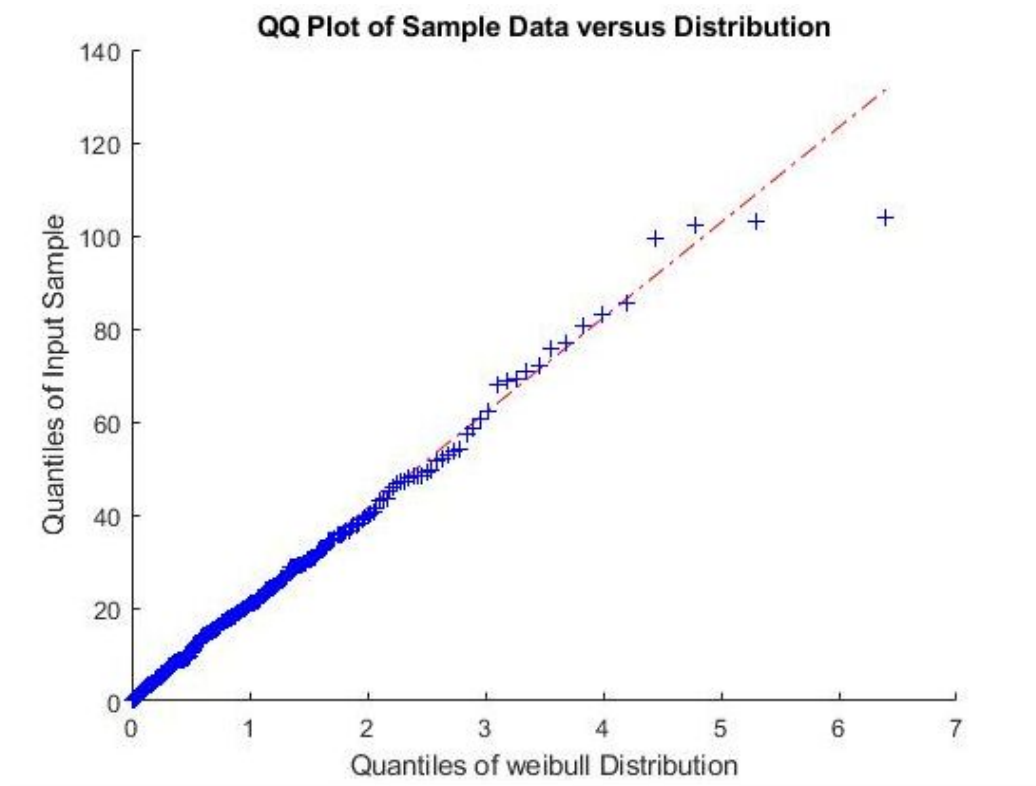


Figure 12: Q-Q plot of Weibull Distribution

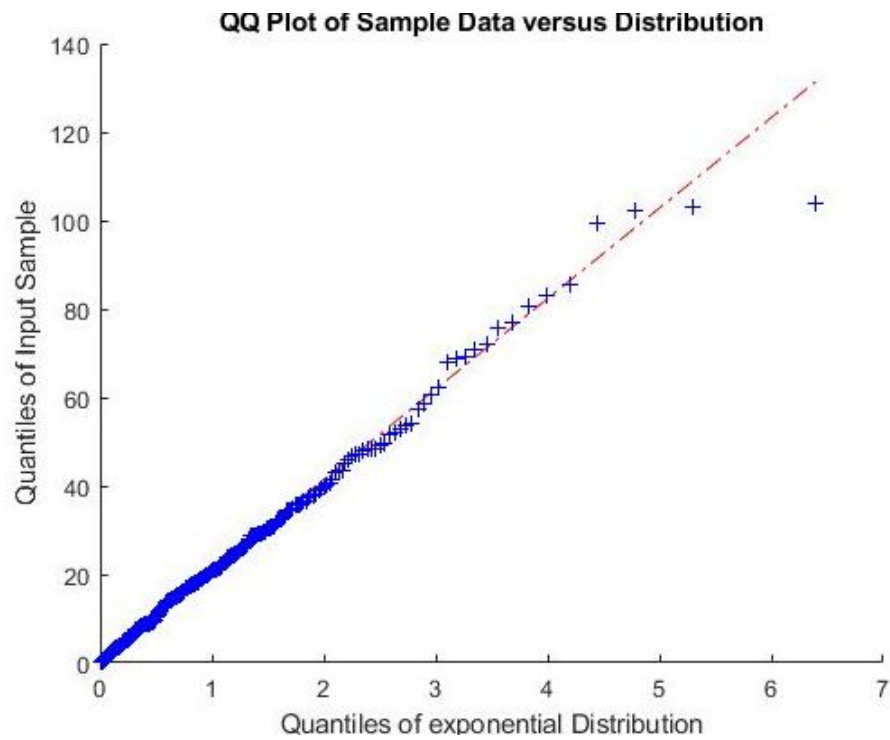


Figure 13: Q-Q plot for Exponential Distribution

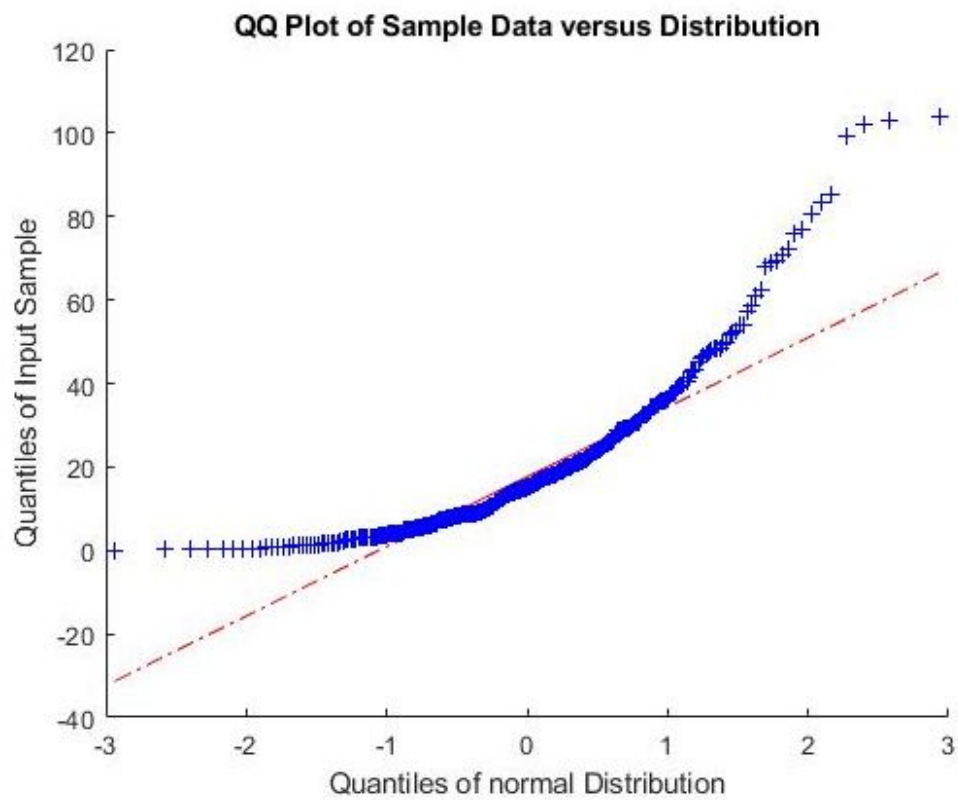


Figure 14: Q-Q plot for Normal Distribution

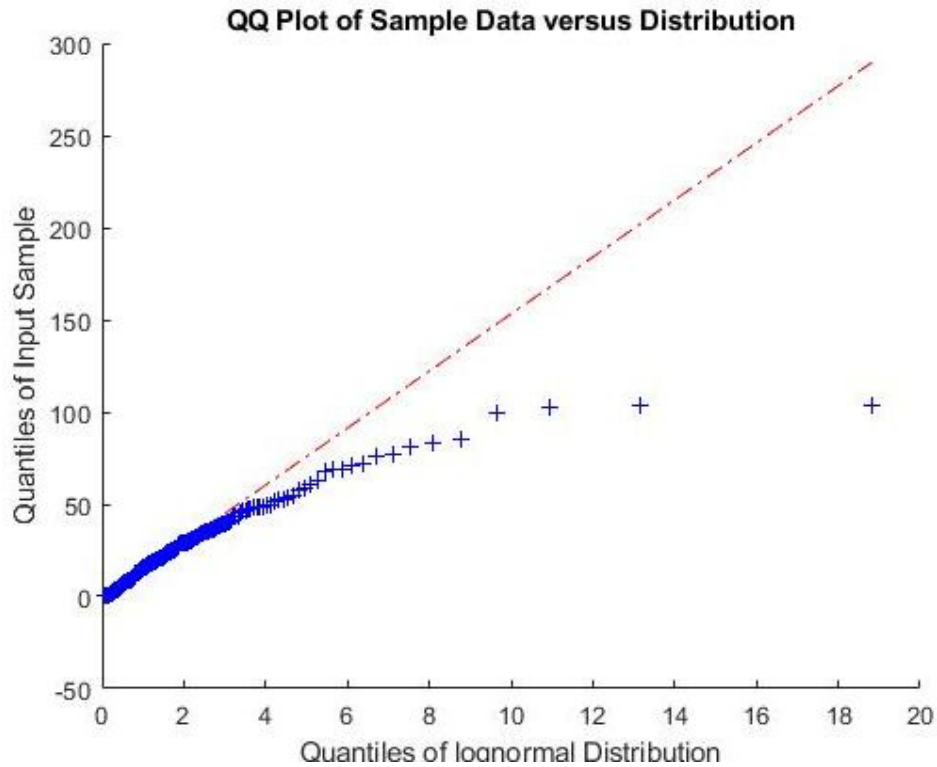


Figure 15: Q-Q plot for Lognormal Distribution

It can be seen above that the exponential distribution and the Weibull distribution both provide a linearity in the Q-Q plot whilst the lognormal and normal are non-linear.

A chi-squared test must now be performed to help with the choosing of the correct distribution. The code below is used to determine if the test passes or not:

```
pd = fitdist(ServiceTimesComponentB, 'Weibull'); %% for chi2gof, distribution can be changed  
chiTest = chi2gof(ServiceTimesComponentB, 'CDF', pd)
```

Whilst running this code, the value for pd is 1 for log and lognormal which means reject the hypothesis for that specific distributions tested. Pd was 0 for exponential and Weibull which means it passes the hypothesis, so both exponential and Weibull could be used to represent the inspection times for component 3, C3.

5.2 Assembly time for Workstations

For workstation 1

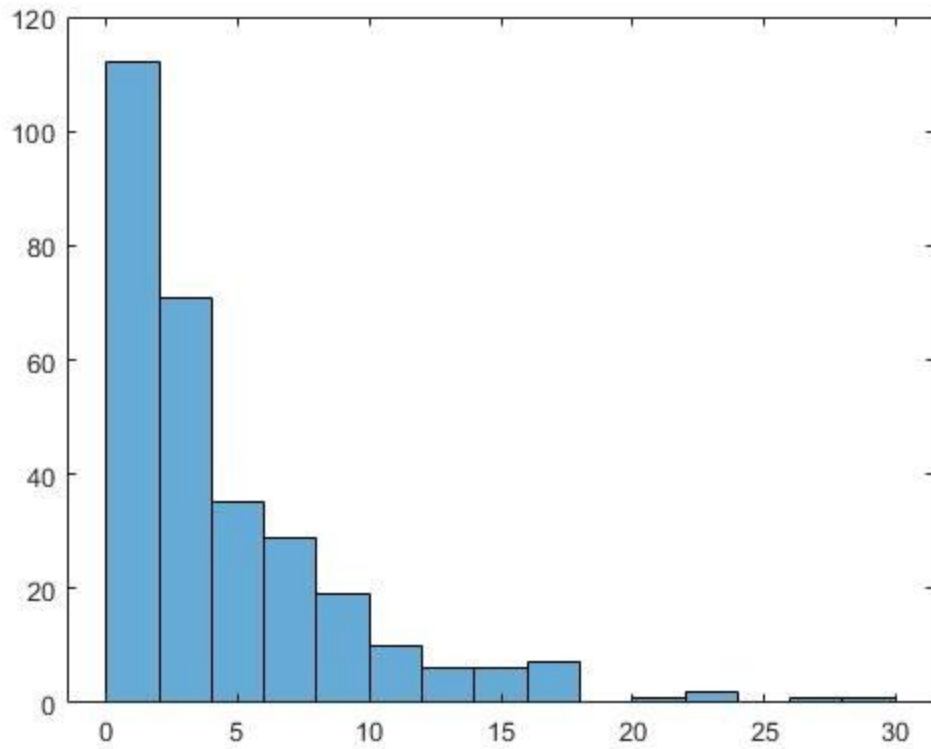


Figure 16. histogram for workstation 1

Based on the shape of figure 1, the data could fit into exponential, weibull or gamma distribution. Therefore, QQ plot and Chi Square test method must be used to finalize the distribution. The QQ plot is a useful tool for evaluating distribution fit, moreover, the Chi Square test is useful for conducting hypothesis tests.

The QQ plot is done By matlab using the code of

```
expDist = makedist('type of the distribution');  
qqplot(ServiceTimesComponentA,expDist);
```

The identified distributions for QQ plot

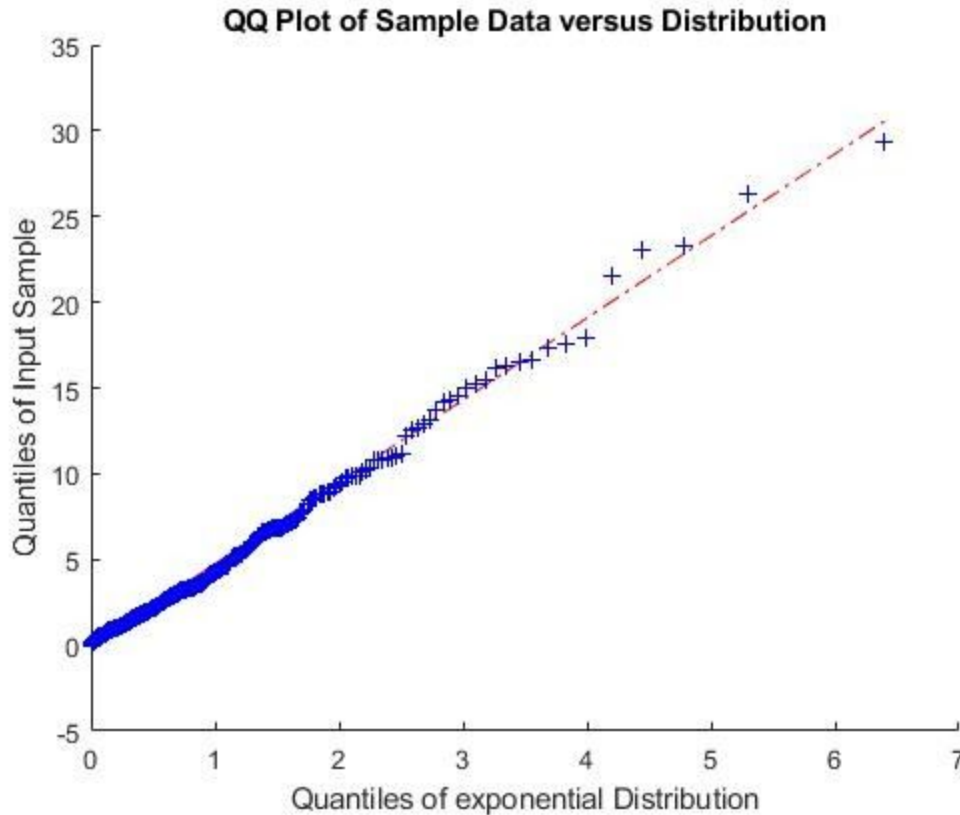


Figure 17. Exponential distribution for QQ plot workstation 1

According to figure 2, the QQ plot for exponential distribution for workstation 1 is a suitable fit. This hypothesis is needed to be proved further by the Chi Square test. Run the ChiTest, the result is 0, which accepts the hypothesis.

```
pd = fitdist(ServiceTimesWorkStationOne, 'Exponential');  
chiTest = chi2gof(ServiceTimesWorkStationOne, 'CDF', pd)
```

The Chi Square test which proves the Exponential distribution hypothesis is true.

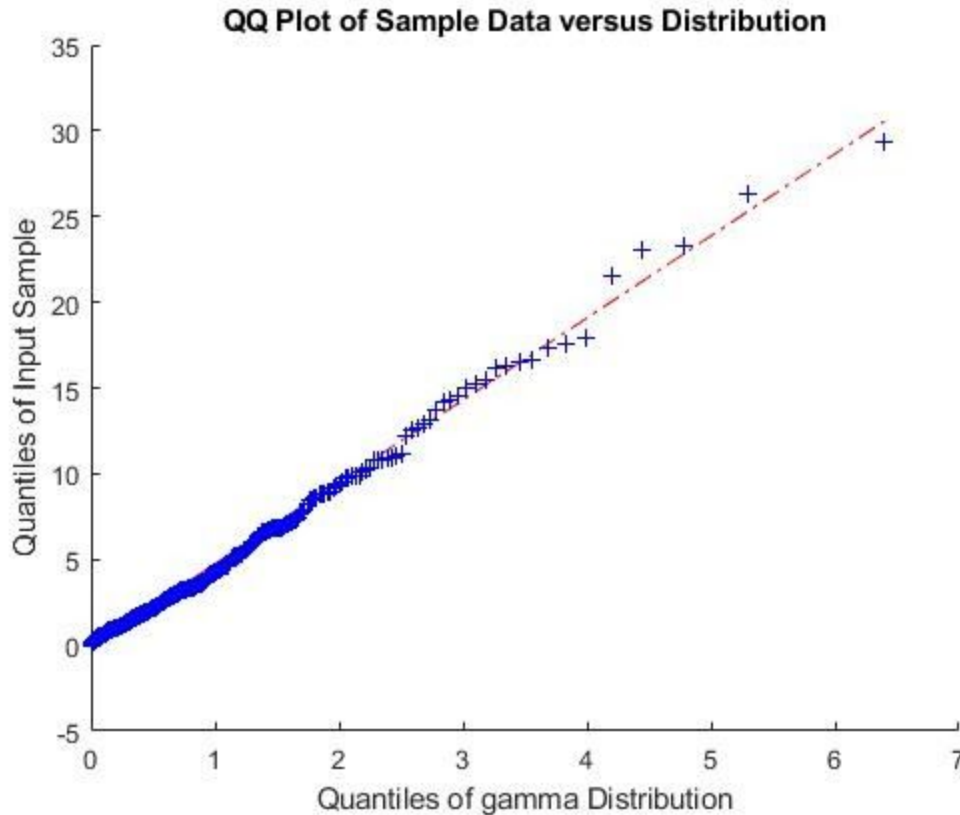


Figure 18. Gamma distribution for QQ plot workstation 1

According to figure 3, the QQ plot for exponential distribution for workstation 1 is a suitable fit. This hypothesis is needed to be proved further by the Chi Square test. Run the ChiTest, the result is 0, which accepts the hypothesis.

```
pd = fitdist(ServiceTimesWorkStationOne, 'Gamma');  
chiTest = chi2gof(ServiceTimesWorkStationOne, 'CDF', pd)
```

The Chi Square test which proves the Gamma distribution hypothesis is true.

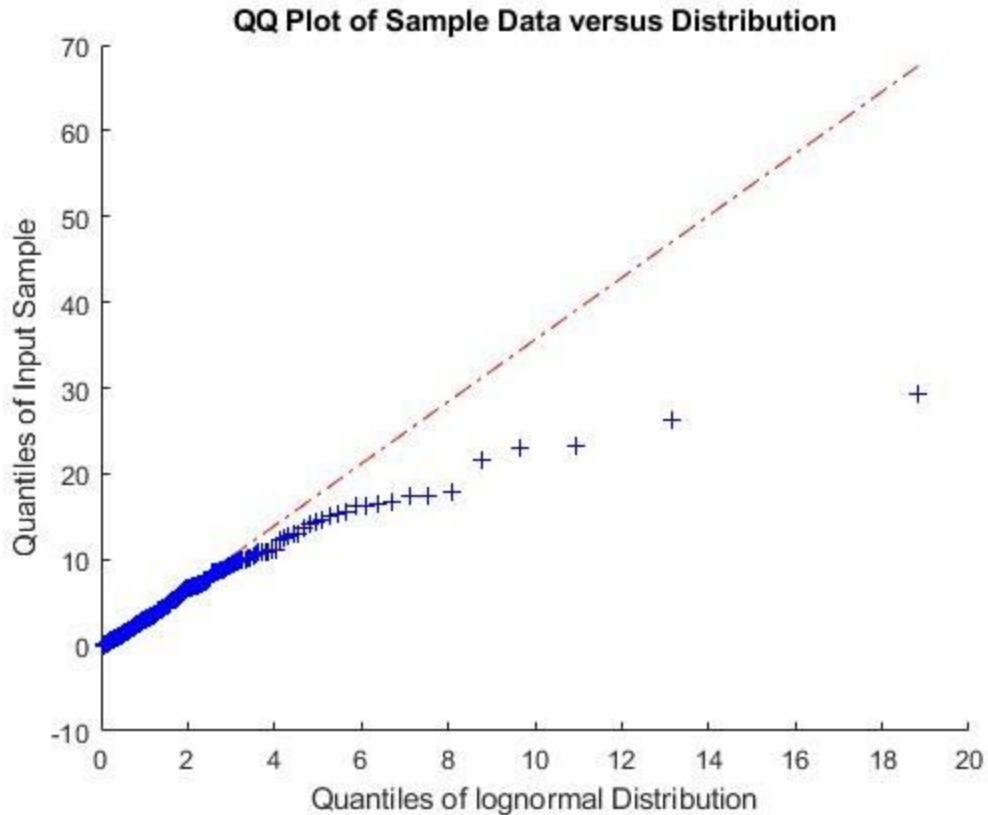


Figure 19. Lognormal distribution for QQ plot workstation 1

According to figure 2, the QQ plot for exponential distribution for workstation 1 is a suitable fit. This hypothesis is needed to be proved further by the Chi Square test. Run the ChiTest, the result is 1, which rejects the hypothesis.

```
pd = fitdist(ServiceTimesWorkStationOne, 'Lognormal');  
chiTest = chi2gof(ServiceTimesWorkStationOne, 'CDF', pd)
```

The Chi Square test which proves the Lognormal distribution hypothesis is false.

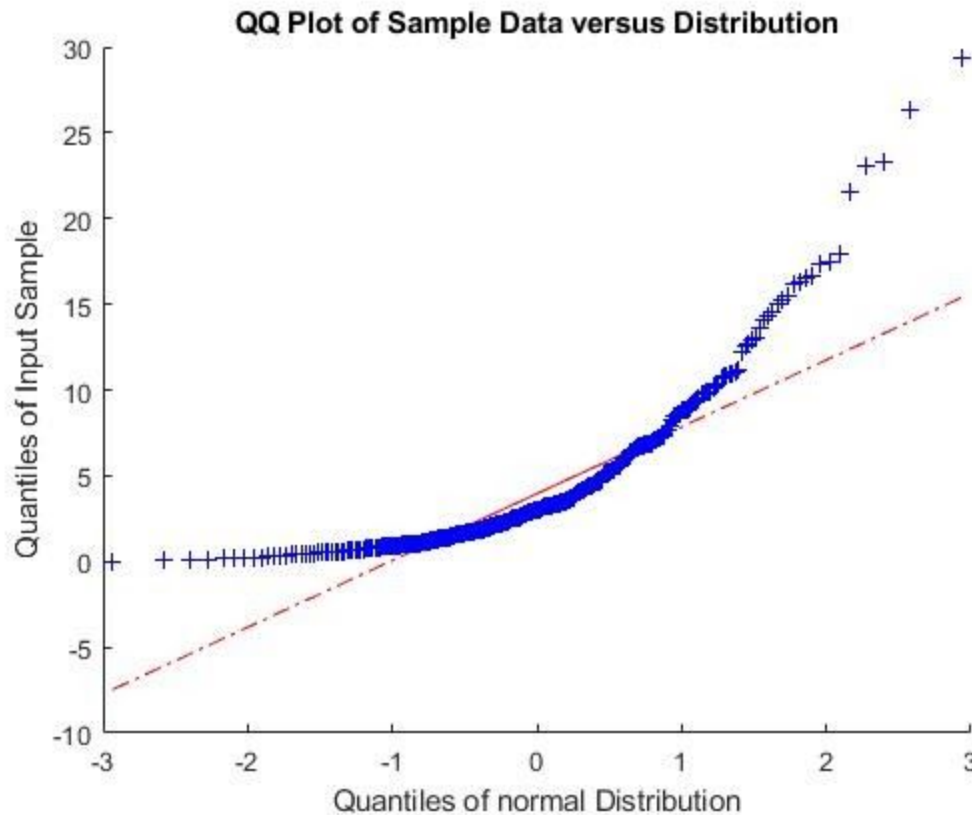


Figure 20. Normal Distribution for QQ plot workstation 1

According to figure 2, the QQ plot for exponential distribution for workstation 1 is a suitable fit. This hypothesis is needed to be proved further by the Chi Square test. Run the ChiTest, the result is 1, which rejects the hypothesis.

```
pd = fitdist(ServiceTimesWorkStationOne, 'Normal');  
chiTest = chi2gof(ServiceTimesWorkStationOne, 'CDF', pd)
```

The Chi Square test which proves the Normal distribution hypothesis is false.

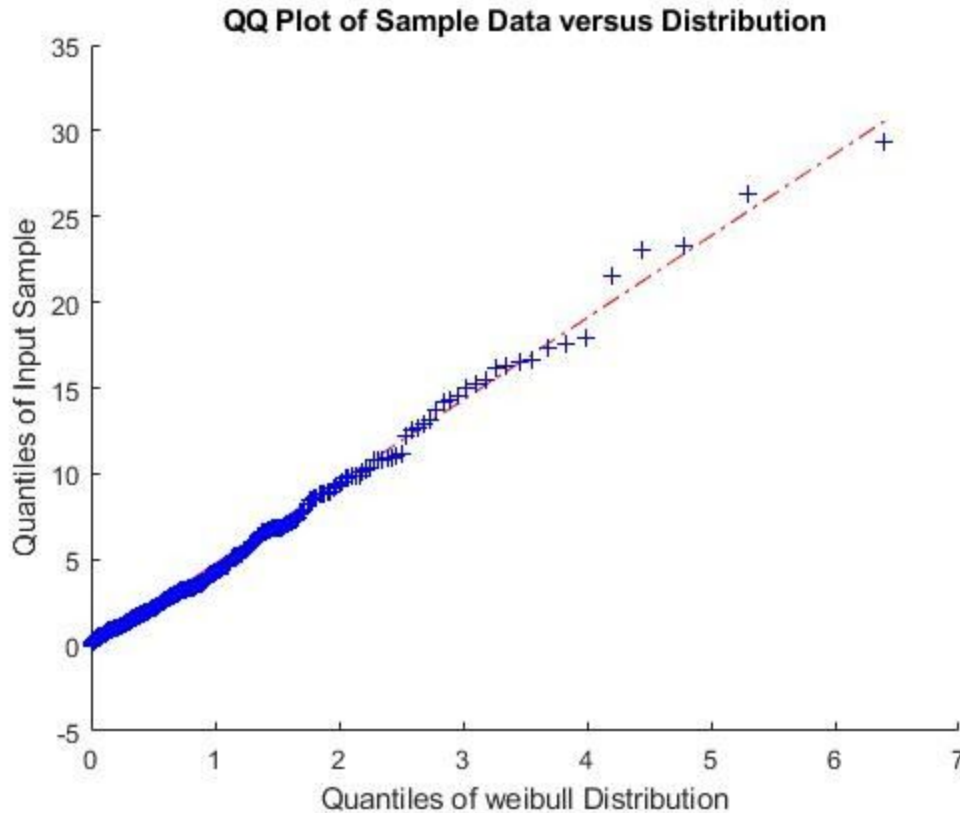


Figure 21. Weibull Distribution for QQ plot workstation 1

According to figure 2, the QQ plot for exponential distribution for workstation 1 is a suitable fit. This hypothesis is needed to be proved further by the Chi Square test. Run the ChiTest, the result is 1, which rejects the hypothesis.

```
pd = fitdist(ServiceTimesWorkStationOne, 'Weibull');  
chiTest = chi2gof(ServiceTimesWorkStationOne, 'CDF', pd)
```

The Chi Square test which proves the Weibull distribution hypothesis is false.

To sum it up, the exponential distribution and the gamma distribution passed the Chi Square test with the value of 0. Furthermore, the QQ plots for the exponential and gamma distributions fits the data set of a straight line. However Gamma distributions are more general than Exponential. In this case, we run another test called the Kolmogorov-Smirnov Test By using this line of code

```
p = kstest(ServiceTimesWorkStationOne, 'CDF', pd)
```

The results show that both parameters are 0. Therefore, exponential distribution and gamma distribution are all good fit for the data set.

For workstation 2

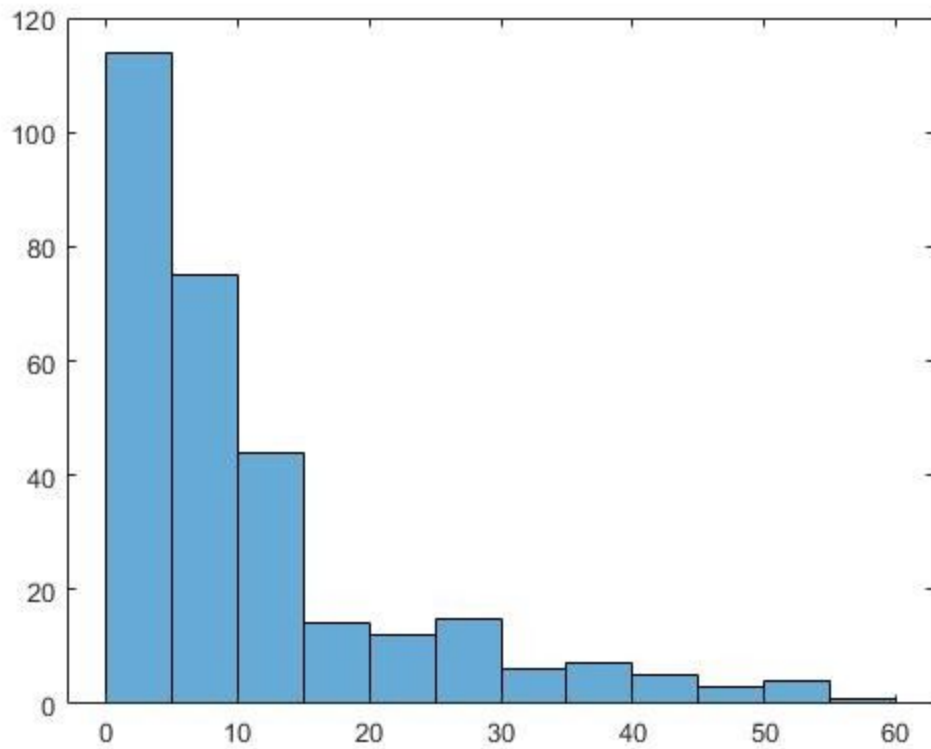


Figure 22. Histogram for workstation 2

According to the shape of figure 7, the data could fit into exponential, weibull or gamma distribution. The QQ plot and Chi Square are used to determine the distribution of the set of data.

The QQ plot is done By matlab using the code of

```
expDist = makedist('type of distribution');  
qqplot(ServiceTimesComponentA,expDist);
```

The identified distributions for QQ plot

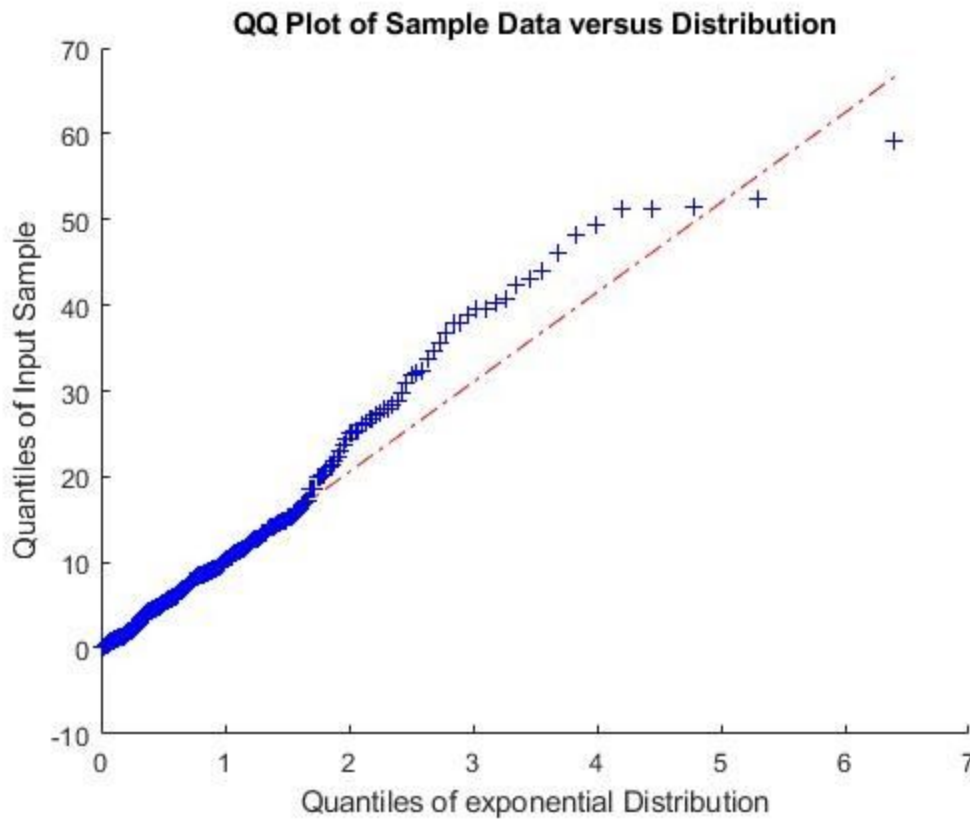
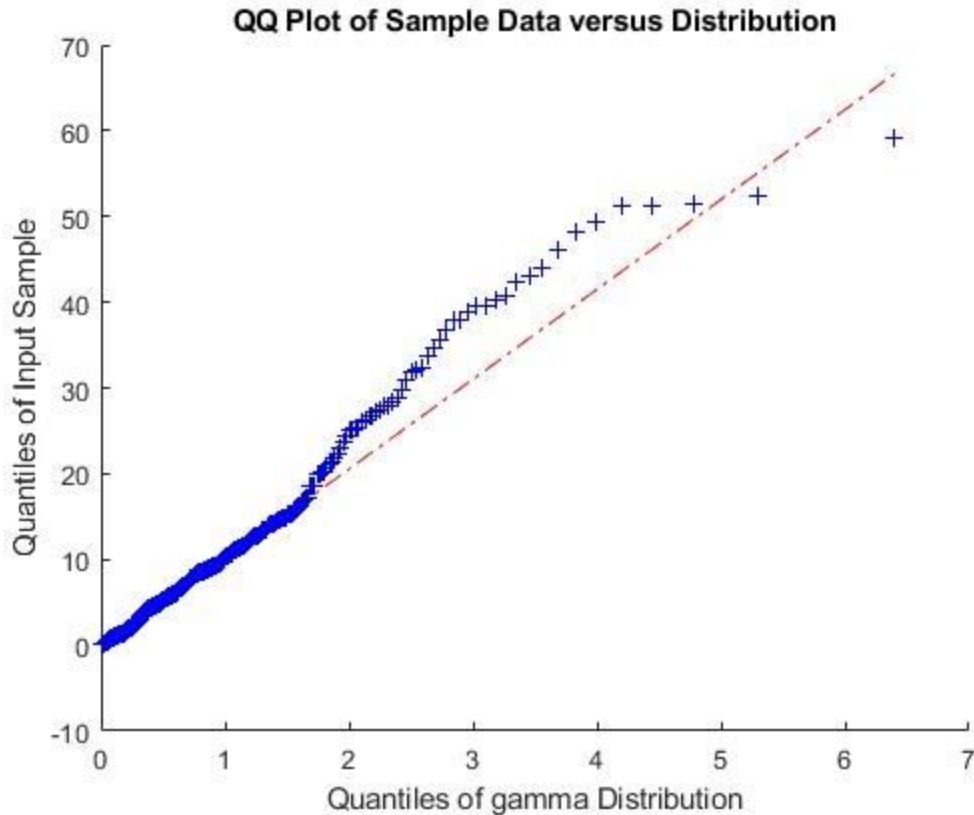


Figure 23. Exponential Distribution for QQ plot workstation 2

According to figure 2, the QQ plot for exponential distribution for workstation 1 is a suitable fit. This hypothesis is needed to be proved further by the Chi Square test. Run the ChiTest, the result is 1, which rejects the hypothesis.

```
pd = fitdist(ServiceTimesWorkStationOne, 'Exponential');  
chiTest = chi2gof(ServiceTimesWorkStationOne, 'CDF', pd)
```

The Chi Square test which proves the Exponential distribution hypothesis is false.



Weibull

Figure24. Gamma Distribution for QQ plot workstation 2

According to figure 2, the QQ plot for exponential distribution for workstation 1 is a suitable fit. This hypothesis is needed to be proved further by the Chi Square test. Run the ChiTest, the result is 0, which accepts the hypothesis.

```
pd = fitdist(ServiceTimesWorkStationOne, 'Gamma');  
chiTest = chi2gof(ServiceTimesWorkStationOne, 'CDF', pd)
```

The Chi Square test which proves the Gamma distribution hypothesis is true.

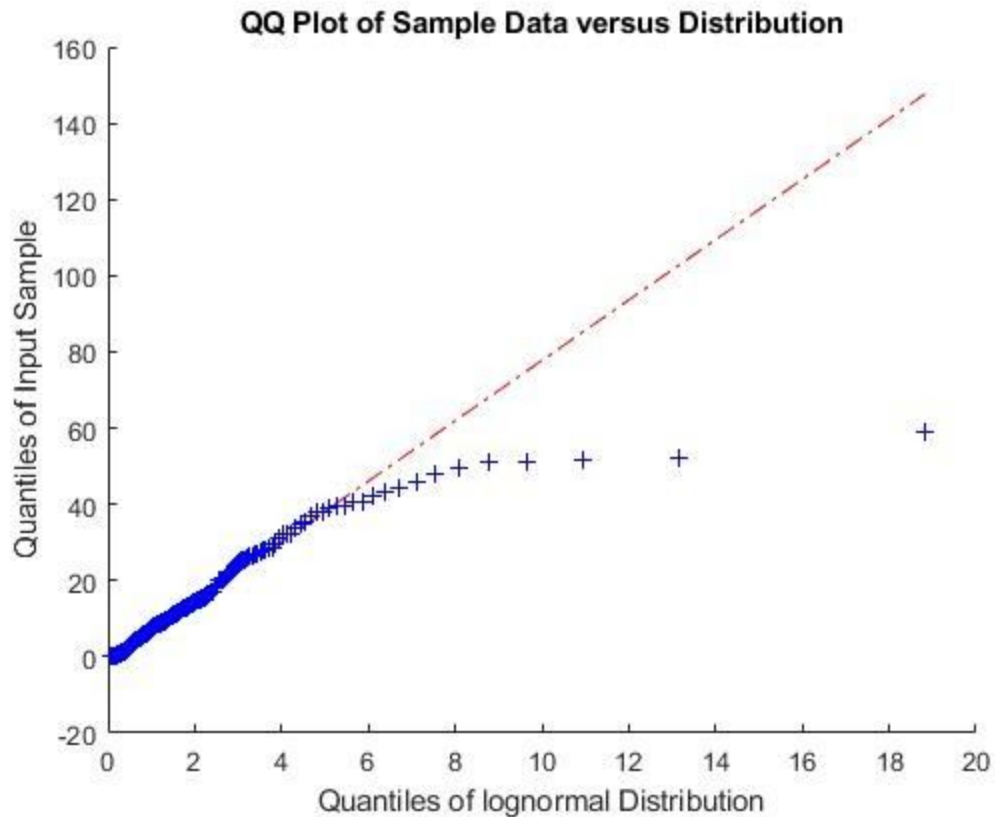


Figure 25. Lognormal Distribution for QQ plot workstation 2

According to figure 2, the QQ plot for exponential distribution for workstation 1 is a suitable fit. This hypothesis is needed to be proved further by the Chi Square test. Run the ChiTest, the result is 1, which rejects the hypothesis.

```
expDist = makedist('Lognormal');  
chiTest = chi2gof(ServiceTimesWorkStationOne, 'CDF', pd)
```

The Chi Square test which proves the Lognormal distribution hypothesis is false.

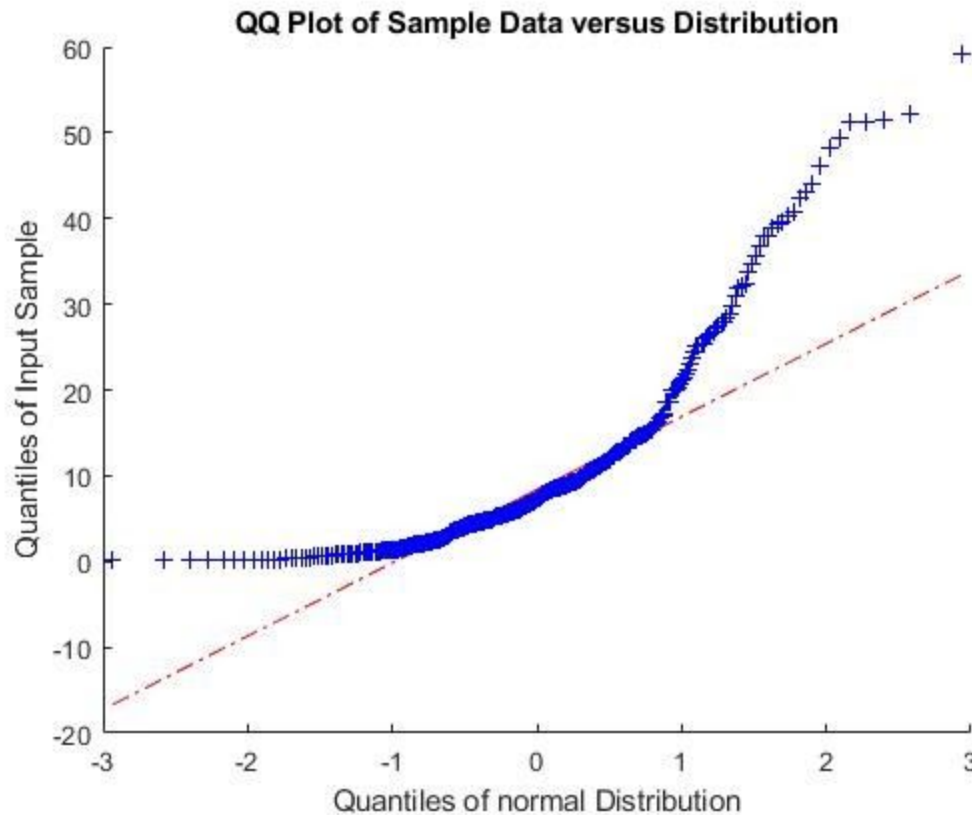


Figure 26. Normal Distribution for QQ plot workstation 2

According to figure 2, the QQ plot for exponential distribution for workstation 1 is a suitable fit. This hypothesis is needed to be proved further by the Chi Square test. Run the ChiTest, the result is 1, which rejects the hypothesis.

```
pd = fitdist(ServiceTimesWorkStationOne, 'Normal');  
chiTest = chi2gof(ServiceTimesWorkStationOne, 'CDF', pd)
```

The Chi Square test which proves the Normal distribution hypothesis is false.

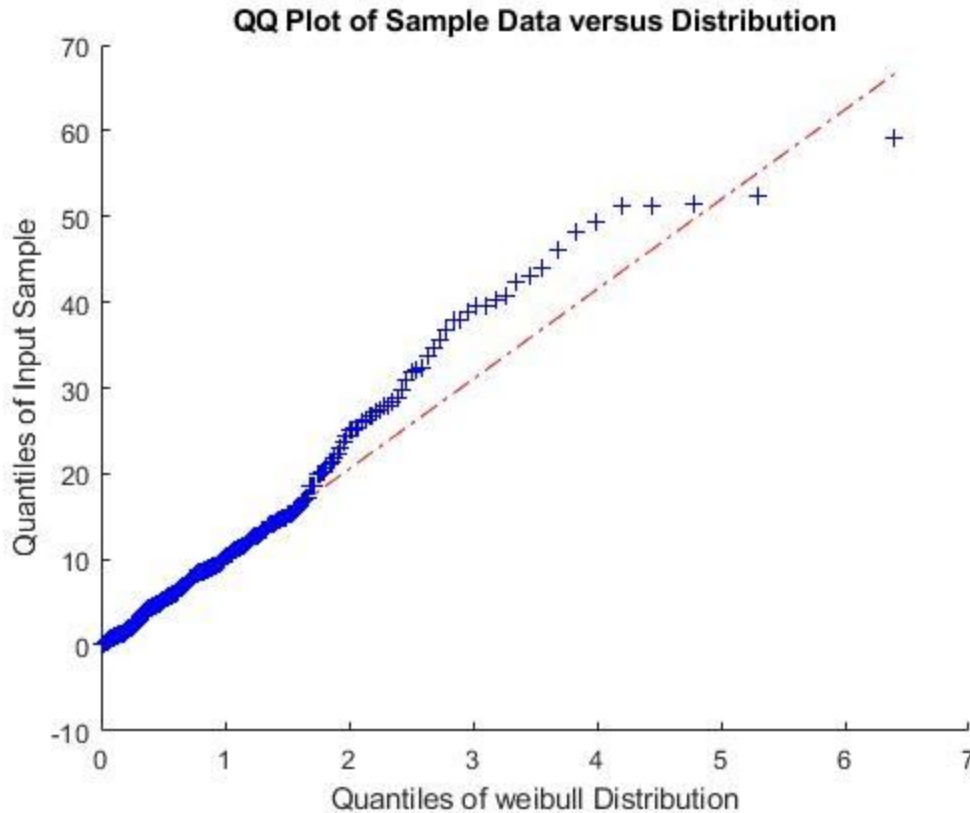


Figure 27. Weibull Distribution for QQ plot workstation 2

According to figure 2, the QQ plot for exponential distribution for workstation 1 is a suitable fit. This hypothesis is needed to be proved further by the Chi Square test. Run the ChiTest, the result is 0, which accepts the hypothesis.

```
pd = fitdist(ServiceTimesWorkStationOne, 'Weibull');  
chiTest = chi2gof(ServiceTimesWorkStationOne, 'CDF', pd)
```

The Chi Square test which proves the Weibull distribution hypothesis is true.

To sum it up, the weibull distribution and the gamma distribution passed the Chi Square test with the value of 0. Furthermore, the QQ plots for the weibull and gamma distributions fits the data set of a straight line.

In this case, we run another test called the Kolmogorov-Smirnov Test By using this line of code

```
p = kstest(ServiceTimesWorkStationOne, 'CDF', pd)
```

The results show that both parameters are 0. Therefore, weibull distribution and gamma distribution are all good fit for the data set.

For workstation 3

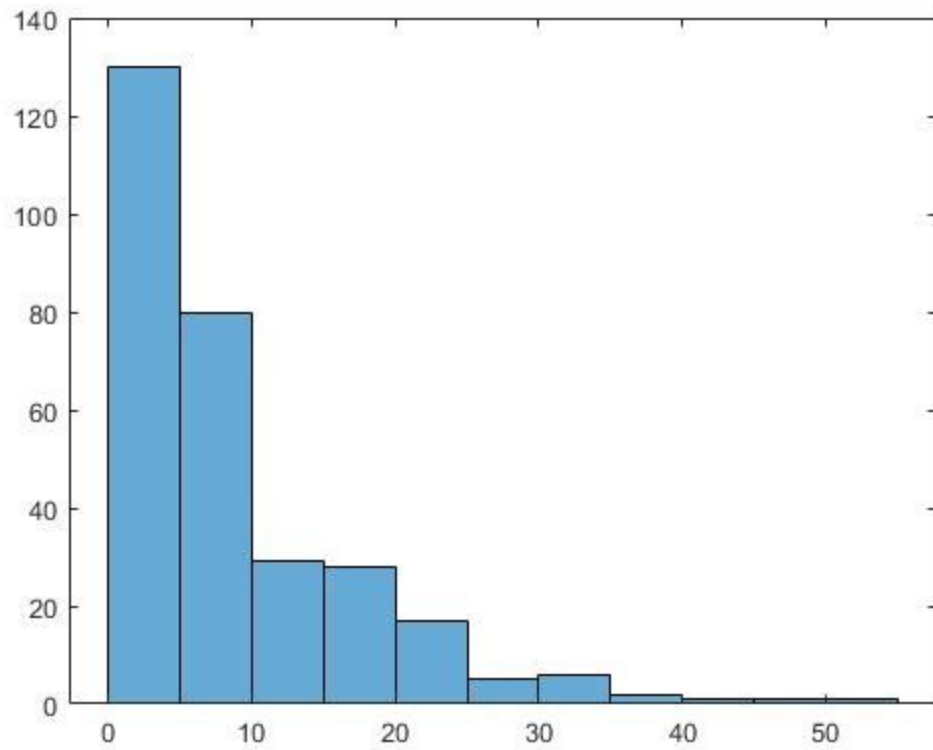


Figure 28. Histogram for workstation 3

According to the shape of figure 7, the data could fit into exponential, weibull or gamma distribution.

The identified distributions for QQ plot

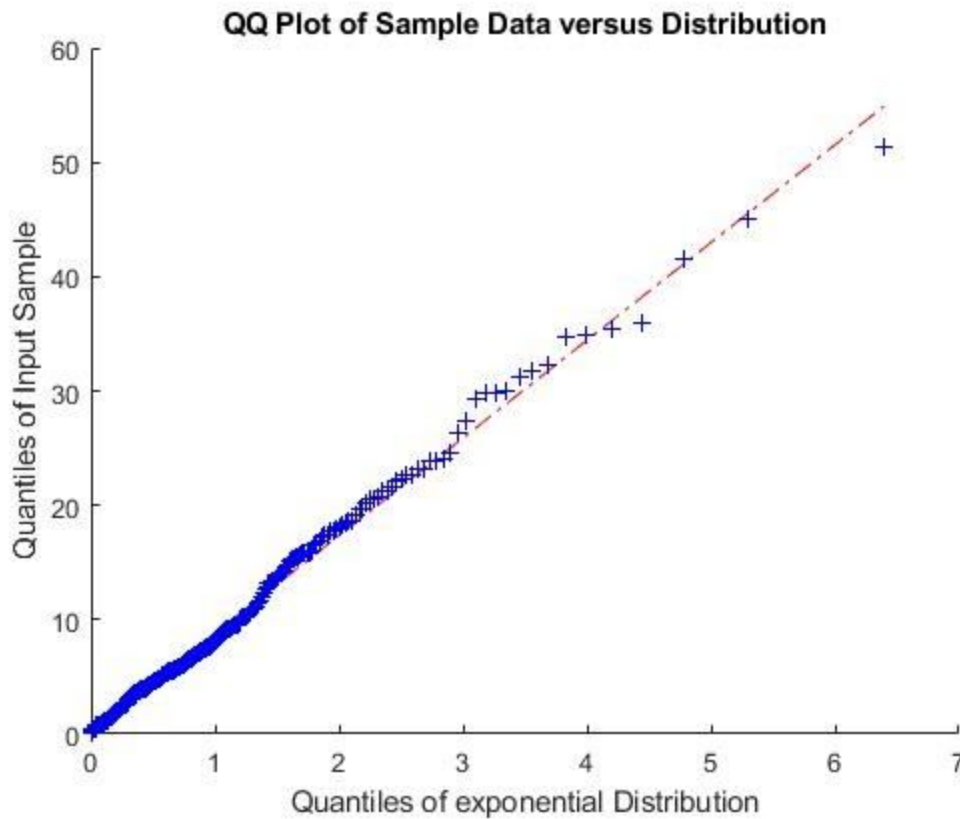


Figure 29. Exponential Distribution for QQ plot workstation 3

According to figure 2, the QQ plot for exponential distribution for workstation 1 is a suitable fit. This hypothesis is needed to be proved further by the Chi Square test. Run the ChiTest, the result is 0, which accepts the hypothesis.

```
pd = fitdist(ServiceTimesWorkStationOne, 'Exponential');  
chiTest = chi2gof(ServiceTimesWorkStationOne, 'CDF', pd)
```

The Chi Square test which proves the Exponential distribution hypothesis is true.

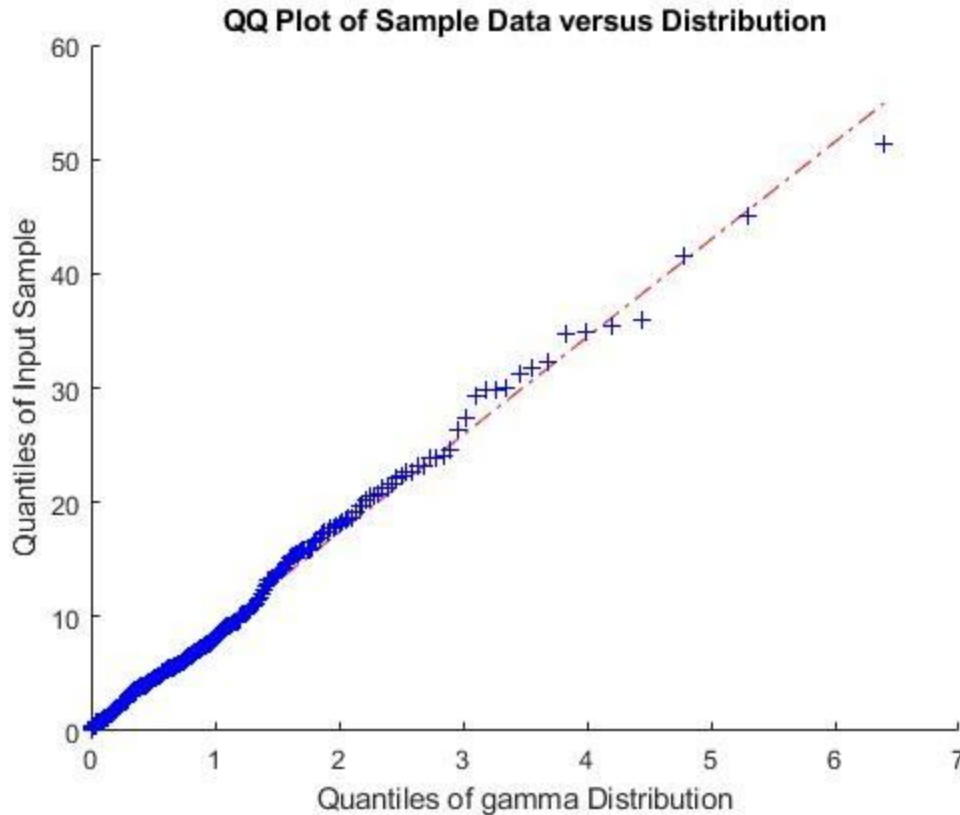


Figure 30. Gamma Distribution for QQ plot workstation 3

According to figure 2, the QQ plot for exponential distribution for workstation 1 is a suitable fit. This hypothesis is needed to be proved further by the Chi Square test. Run the ChiTest, the result is 0, which accepts the hypothesis.

```
pd = fitdist(ServiceTimesWorkStationOne, 'Gamma');  
chiTest = chi2gof(ServiceTimesWorkStationOne, 'CDF', pd)
```

The Chi Square test which proves the Gamma distribution hypothesis is true.

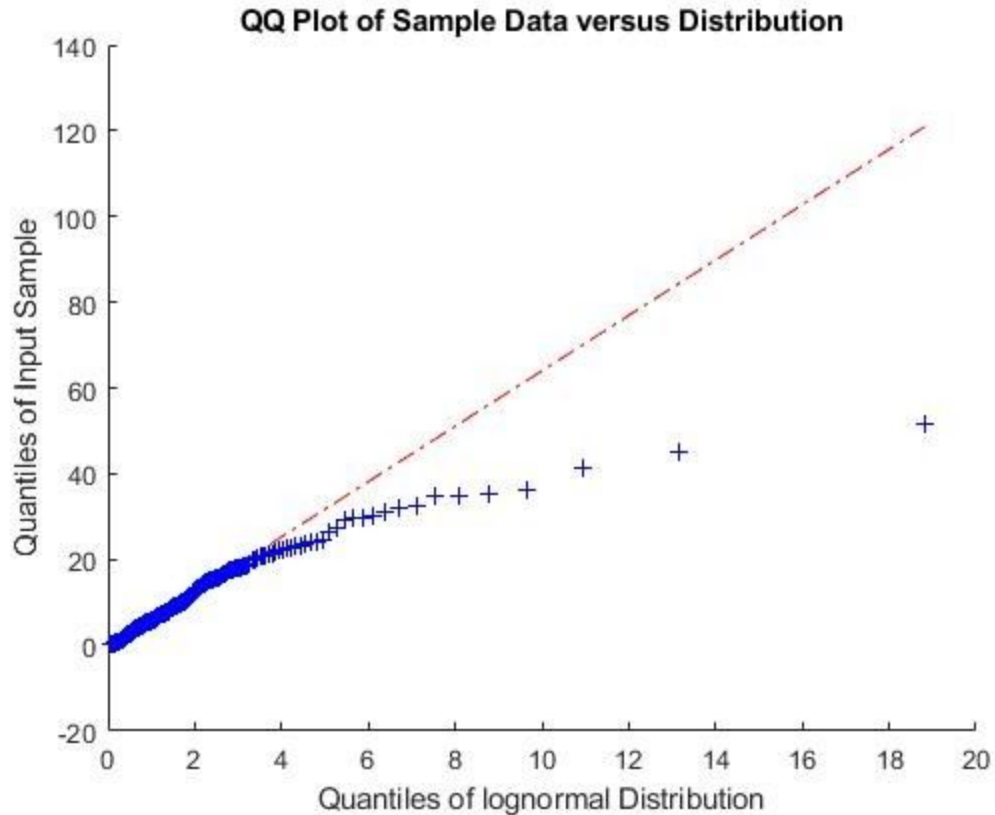


Figure 31. Lognormal Distribution for QQ plot workstation 3

According to figure 2, the QQ plot for exponential distribution for workstation 1 is a suitable fit. This hypothesis is needed to be proved further by the Chi Square test. Run the ChiTest, the result is 1, which rejects the hypothesis.

```
pd = fitdist(ServiceTimesWorkStationOne, 'Lognormal');  
chiTest = chi2gof(ServiceTimesWorkStationOne, 'CDF', pd)
```

The Chi Square test which proves the Lognormal distribution hypothesis is false.

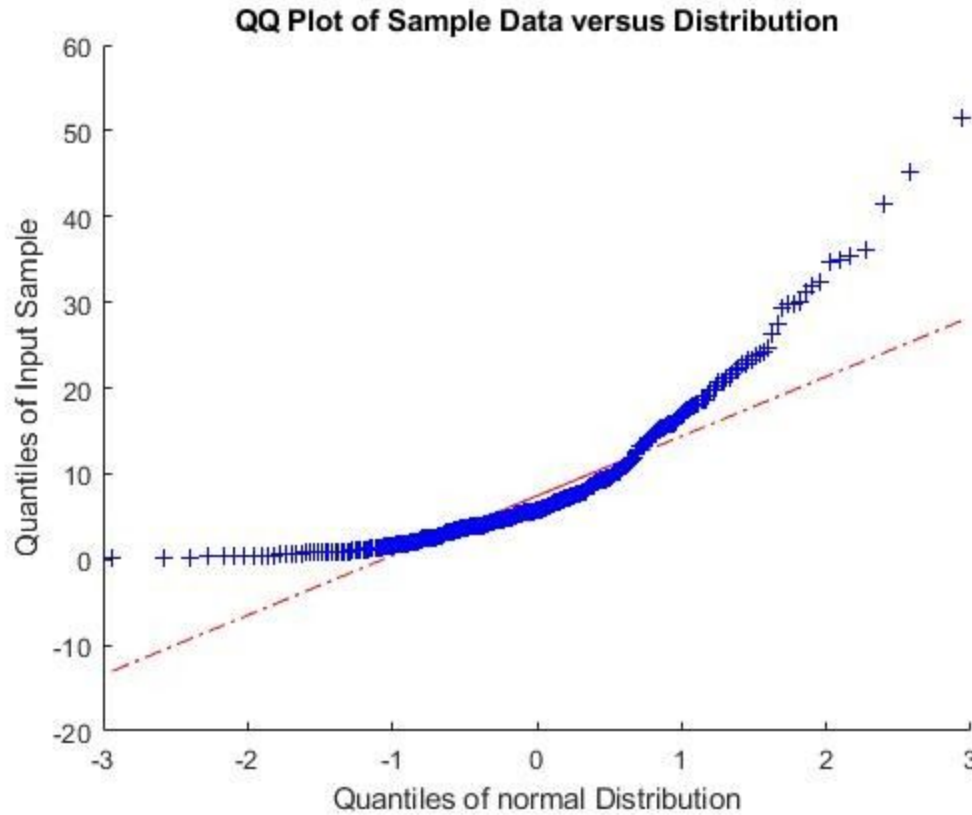


Figure 32. Normal Distribution for QQ plot workstation 3

According to figure 2, the QQ plot for exponential distribution for workstation 1 is a suitable fit. This hypothesis is needed to be proved further by the Chi Square test. Run the ChiTest, the result is 1, which rejects the hypothesis.

```
pd = fitdist(ServiceTimesWorkStationOne, 'Normal');  
chiTest = chi2gof(ServiceTimesWorkStationOne, 'CDF', pd)
```

The Chi Square test which proves the Normal distribution hypothesis is false.

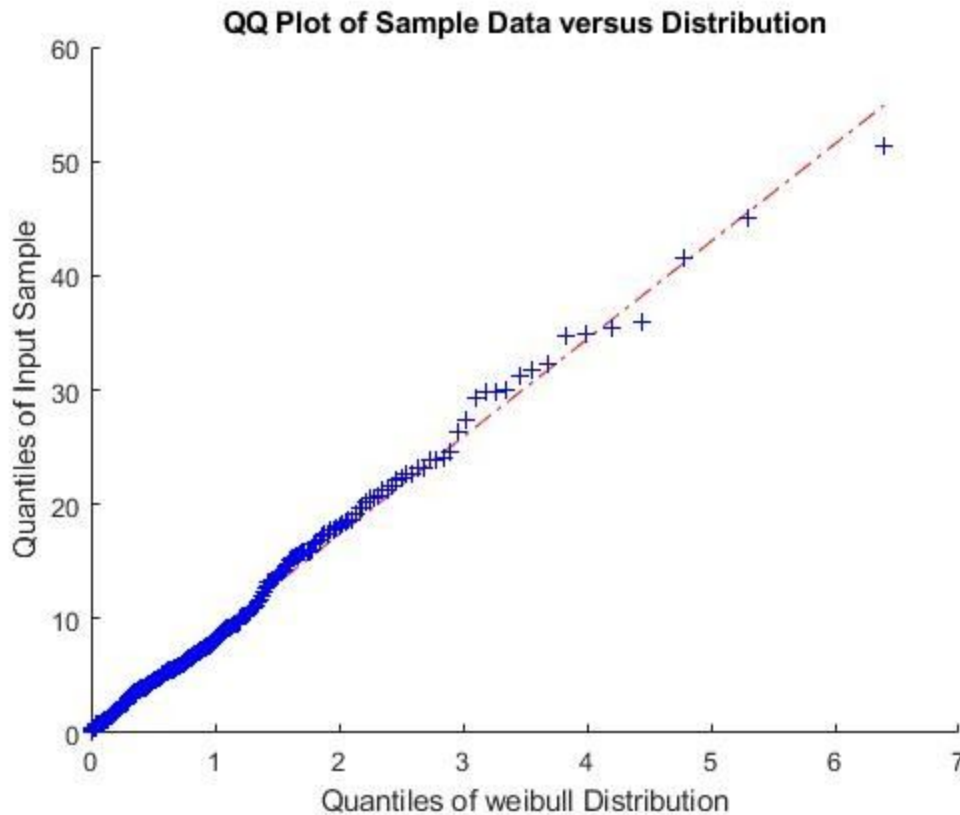


Figure 33. Weibull Distribution for QQ plot workstation 3

According to figure 2, the QQ plot for exponential distribution for workstation 1 is a suitable fit. This hypothesis is needed to be proved further by the Chi Square test. Run the ChiTest, the result is 0, which accepts the hypothesis.

```
pd = fitdist(ServiceTimesWorkStationOne, 'Weibull');  
chiTest = chi2gof(ServiceTimesWorkStationOne, 'CDF', pd)
```

The Chi Square test which proves the Weibull distribution hypothesis is true.

To sum it up, the weibull distribution, gamma distribution and exponential distribution passed the Chi Square test with the value of 0. Furthermore, the QQ plots for the weibull and gamma distributions fits the data set of a straight line.

In this case, we run another test called the Kolmogorov-Smirnov Test By using this line of code

```
p = kstest(ServiceTimesWorkStationOne, 'CDF', pd)
```

The results show that both parameters are 0. Therefore, all the three distribution -- weibull distribution, gamma distribution and exponential distribution are all good fit for the data set.

6.0 Input Modeling Part 2 - Input Generation

In section 5.0, it was determined that not all data sets were a good fit for an exponential distribution. Weibull and Gamma Distributions were both good fits for our data sets but since gamma distributions were not covered in the course slides, we've decided to adopt Weibull distributions as the best fitting distributions of our historical data set.

Since we've determined that the historical data sets follow weibull distributions. The next step is then to determine how to generate weibull distribution random numbers and to make sure these numbers would reflect historical data for our factory model. We've accomplished this through the use of matlab, to determine how to generate the weibull random numbers, and java, to actually generate trials of these random numbers.

6.1 Determining how to generate useful data using MATLAB

Generating similar weibull distributions, reflective of the historical data set provided, can be done if we know the alpha(scale) and beta(shape) of the weibull distributions. Fortunately matlab allows us to use the command `[parmhat] = wblfit(data)`; which examines a dataset and extrapolates the alpha(scale) and beta(shape) coefficients with a 95% confidence value.

Table 1 below, you will see the code required to read the workstation3 service times historical data file "ws3.dat" and then extract the alpha and beta parameters. I've also included comments of the coefficient results for all other workstations and components.

Table1: matlab code to extract alpha and beta coefficients from the historical data files.

```
%%open each historical data file and extract its contents.
fileID = fopen('ws3.dat','r');
[data]=fscanf(fileID,'%f');
fclose(fileID);
[parmhat] = wblfit(data); %gets the alpha and beta coefficients from our distributions.







%C1 - [10.7250489541748,1.09600229615940]
%C2 - [16.0876383178756,1.09600766386768]
%C3 - [20.9019593805947,1.03269263226412]
%WS1 - [4.65209941000056,1.02396073105742]
%WS2 - [10.6815122330479,0.922808316137963]
%WS3 - [8.98290847458538,1.05214687927421]
```

Now that we know the coefficients of the weibull distributions from the historical data files, we can use java and some external libraries to assist us in generating useful input comparable to our historical data files.

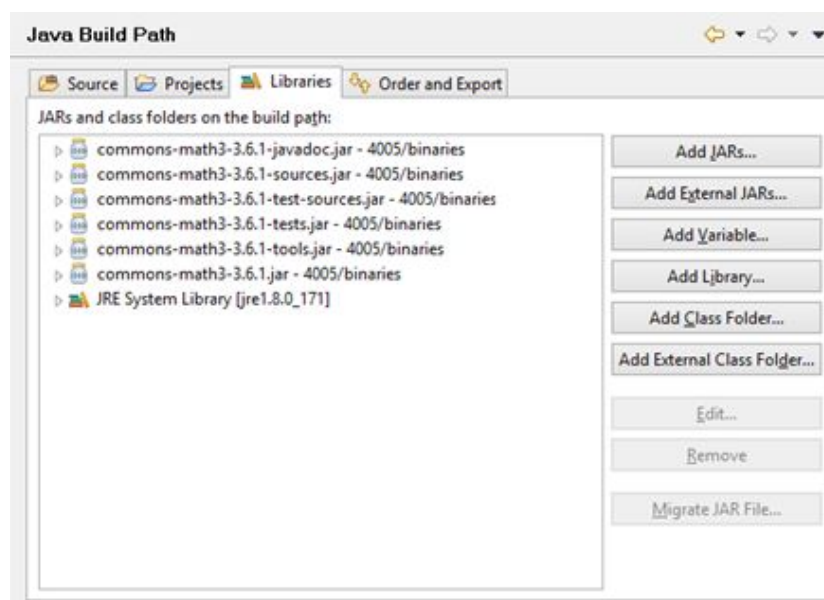
6.2 Using Java To Generate Random Numbers Based On The Weibull Distributions From The Historical Data Files

While it would be difficult to write a weibull distribution random number generator by ourselves, fortunately there are many tools available for us so we don't have to reinvent the wheel. We can use the built in weibull distribution generators provided in the Apache-commons-math binaries to make the generation of reliable and useful random numbers reflective of the historical data weibull distributions very easy.

You can find the Apache-commons-math binaries used in the “binaries folder” in our deliverable II submission folder.

OS_Install (C:) > Users > Jaycaybe > Downloads > 4005-master.zip > 4005-master > binaries				
Name	Type	Compressed size	Pass	
 commons-math3-3.6.1.jar	Executable Jar File	1,920 KB	No	
 commons-math3-3.6.1-javadoc.jar	Executable Jar File	6,241 KB	No	
 commons-math3-3.6.1-sources.jar	Executable Jar File	2,285 KB	No	
 commons-math3-3.6.1-tests.jar	Executable Jar File	2,642 KB	No	
 commons-math3-3.6.1-test-sourc...	Executable Jar File	1,783 KB	No	
 commons-math3-3.6.1-tools.jar	Executable Jar File	11 KB	No	

If you would like to run our simulation, please remember to include the binaries to the Java build path, since the “generateWeibullDistributionData.java” class i will talk about later uses weibull tools from these binaries. If you are using eclipse, this is done by right clicking the project, going into properties and selecting the Java build path and then adding the binaries.



6.2.1 The “generateWeibullDistributionData.java” class

The weibull distribution random number generator is done via the addition of this class. In this class we do the following:

- Using the alpha(scale) and beta(shape) coefficients from each historical data file (shown in the table below but also mentioned in section 6.1) we generate a weibull distribution.
- Because, as mentioned before, the weibull distribution generation is done via the alpha and beta coefficients from the historical data, we can then sample these generated weibull distributions 300 times at random, for each data set, for very reliable simulated input data.
- We then “currently” record these results in 6 files called “C1.txt,C2.txt,C3.txt,WS1.txt,WS2.txt,WS3.txt” which can be found in the trial4Data folder.

Table 2: Alpha and Beta coefficients from our historical data files

For	alpha(scale)	beta(shape)
Component A	10.7250489541748	1.09600229615940
Component B	16.0876383178756	1.09600766386768
Component C	20.9019593805947	1.03269263226412
WorkStation#1	4.65209941000056	1.02396073105742
WorkStation#2	10.6815122330479	0.922808316137963
WorkStation#3	8.98290847458538	1.05214687927421

The actual piece of code used to generate the weibull distribution and get a random numbers is as follows:

```
public static double WeibullDistr(double alpha, double beta) {  
    return new WeibullDistribution(alpha, beta).sample();  
}
```

The .sample() method from the “Apache-commons-math binaries” is extremely effective. It will return a single random value from a weibull distribution. So far we have NEVER sampled a duplicate value, as you can see from the excel file provided.

6.2.2 The “SimulateFactoryModel2.java” class

This class is identical to the one from deliverable I. The only change is that it now calls the 6 files called “C1.txt,C2.txt,C3.txt,WS1.txt,WS2.txt,WS3.txt” which can be found in the trial4Data folder. Each file contains 300 randomly generated service times.

6.3 Justifying why we believe our generated data (input data) is reliable and reflective of the real world

When generating our “input data”, I decided to record 4 trials for component A/Component 1.

Since all numbers are generated via the **exact same method** of using the historical data’s weibull alpha(scale) and beta(shape) coefficients, if i prove that we generate good service time (input data) for component1/A, then i’ve proven that all of our input data is generated correctly.

In a file called” Deliverable2 - mean analysis of the 4 trials of generated input for the component A”. Inserting a table of size 300 would make this document very messy (i tried). Instead i recommend you reference that excel file if you want to see our generated input for component A, however the results are summarized in the table below.

Table3: Mean Results of Generated Trials (300 service times per trial) for Component One

For Component One	Mean
Trial 1	10.51645611
Trial 2	10.14726936
Trial 3	10.79657034
Trial 4	9.541369146
Historical Data (sirvinsp1.dat)	10.35791

This table shows the mean of each trial , when generating 300 input data for component A. From this table,

- The average mean of the 4 trials was: 10.25042.
- $10.25042 / 10.35791(\text{historical mean for component A}) = 98.96220606 \%$
- This means that over 4 trials, the average mean was only 1.037793938% off from the historical data mean.

I believe that if we were to run more and more trials, say n trials, as n approaches infinity, we would be infinitesimally close to the historical data mean.

Once again, this is because we used MATLAB to extract the α (scale) and β (shape) coefficients of the weibull distributions from each historical data file.

6.4 - Running our simulation using trial 4 generated input data (you can find this in trial4Data folder)

6.4.1 Simulation Results From Deliverable One (also found on Page 10)

```
The Object: [Inspector#1]
Preformed this many component inspections: [300]
Were RESTRICTED from working due to the buffers(queue) being full for this long: [0.000] minutes
The Inspectors ACTIVE for a total of: [3107.381] minutes
Were Idle for: [%: 0.000] of the total time

The Object: [Inspector#2]
Preformed this many component inspections: [79]
Were RESTRICTED from working due to the buffers(queue) being full for this long: [1593.359] minutes
The Inspectors ACTIVE for a total of: [3121.537] minutes
Were Idle for: [%: 51.044] of the total time

The Object: [WorkStation#1]
Produced a total of: [222] products.
With an avarage production of: [4.27] Products Made/hour

The Object: [WorkStation#2]
Produced a total of: [46] products.
With an avarage production of: [0.88] Products Made/hour

The Object: [WorkStation#3]
Produced a total of: [31] products.
With an avarage production of: [0.60] Products Made/hour
```

6.4.2 Simulation results when using our generated input (found in the trial4Data folder)

```
The Object: [Inspector#1]
Preformed this many component inspections: [300]
Was WAITING from working due to the buffers(queue) being full for this long: [0.000] minutes
The Inspectors ACTIVE for a total of: [2862.418] minutes
Was Idle for: [%: 0.000] of the total time

The Object: [Inspector#2]
Preformed this many component inspections: [86]
Was WAITING from working due to the buffers(queue) being full for this long: [1385.876] minutes
The Inspectors ACTIVE for a total of: [2868.931] minutes
Was Idle for: [%: 48.306] of the total time

The Object: [WorkStation#1]
Produced a total of: [216] products.
With an avarage production of: [4.52] Products Made/hour

The Object: [WorkStation#2]
Produced a total of: [45] products.
With an avarage production of: [0.94] Products Made/hour

The Object: [WorkStation#3]
Produced a total of: [39] products.
With an avarage production of: [0.82] Products Made/hour
```


6.4.3 Comparing Simulation Results - Historical Data vs Generated Data.

The table below compares attributes of our simulation results when using Historical Data vs Our Generated Input.

Table4: Summarization of Historical vs Generated Simulation Results For Inspectors

	Components Inspected Historical	Components Inspected Generated	Active (minutes) Historical	Active (minutes) Generated
Inspector 1	300	300	3107.381	2862.418
Inspector 2	79	86	3121.537	2868.931

Table5: Summarization of Historical vs Generated Simulation Results For Inspectors

	Idle (minutes) Historical	Idle (minutes) Generated	Idle % Historical	Idle % Generated
Inspector 1	0.000	0.000	0.000%	0.000%
Inspector 2	1593.359	1385.876	51.044%	48.307%

Table6: Summarization of Historical vs Generated Simulation Results For Workstations

	Products Produced Historical	Products Produced Generated	Products (made/hour) Generated	Products (made/hour) Generated
WorkStation 1	222	216	4.27	4.52
WorkStation 2	46	45	0.88	0.94
WorkStation 3	31	39	0.60	0.82

6.6.4 - Simulation Result Conclusions

Comparing the Simulations conducted via historical data vs generated (trial4) data, it was found that,

- Inspector 1 results remained the same
- Inspector 2's idle time was about 3% less but inspected on 7 more components
- Inspector 2's results are very similar
- Workstation 1 results were very similar, 222 products produced to the 216 from our generated input as well as similar products made/hour rates
- Workstation 2 results were very similar, 46 products produced to the 45 from our generated input as well as similar products made/hour rates
- Workstation 3 results were less similar, 31 products vs 39 from our generated input as well as a 0.22 products made/hour rate difference.

For inspectors, the values from table 4 and 5 were similar. If I took the mean of all 4 of our trials, we would have been closer to the historical values. I believe that we are simulating the service times for inspectors correctly.

For WorkStations 1-2 the simulation results when using our generated input were extremely close to our simulation when using historical data. WorkStation 3 did have a notable gap but this was one trial, and since inspector 2's policy is to choose what component to inspect at random, this outcome is not unexpected. I believe that if we included the mean of results from all 4 trials, we would have been even closer.

In conclusion i believe the input we generate is representative of the real world.