**Output Data Analysis**

Table of Contents

[Transient and Steady-State Behaviour 3](#_Toc87178249)

[Types of Simulations 5](#_Toc87178250)

[Statistical Analysis for Terminating Simulations 6](#_Toc87178251)

[Estimating Parameters 6](#_Toc87178252)

[Confidence Intervals 7](#_Toc87178253)

[Statistical Analysis for Steady-State Parameters 9](#_Toc87178254)

[Welch Procedure 9](#_Toc87178255)

[Paired Confidence Interval 12](#_Toc87178256)

[Modified Two-Sample Confidence Interval 13](#_Toc87178257)

[Steady-State Performance 14](#_Toc87178258)

The output data we get from a simulation cannot be used directly, since it is possible that the data will **vary greatly** from one run of the simulation to another. To deal with this, we run the program multiple times to get a large amount of output data and then **analyse** the data to determine a fairly accurate approximation for the output.

One of the biggest problems with general users of simulation programs is that they take data from a **single run** of the simulation, which may produce very inaccurate results. The input data for the simulation are generated from **random variates**. This means that the output is a function of those random variates, which causes this issue.

Normally, we simply **average** the output data. For example, if we have a simulation that gives us the average delay in a queue, we run the simulation times and then calculate the average of the averages. In some cases however, this will give us **inaccurate results**. We are assuming that the output data are IID random variables, but they may not truly be **independent**. For example, the delay in a queue is most likely not independent, since the delay faced by one customer depends on the length of the queue, which relates the delays faced by consecutive customers.

To deal with this issue, one thing we could do is to take the average of the delays faced by the first customer from every run, then the average of the delays faced by the second customer and so on and average that instead. Since we are changing the **seed value** at the start of every run, these averages will be IID random values. However, this is a more sophisticated approach. The general approach is still the simpler one, with the average of the averages being taken.

## Transient and Steady-State Behaviour

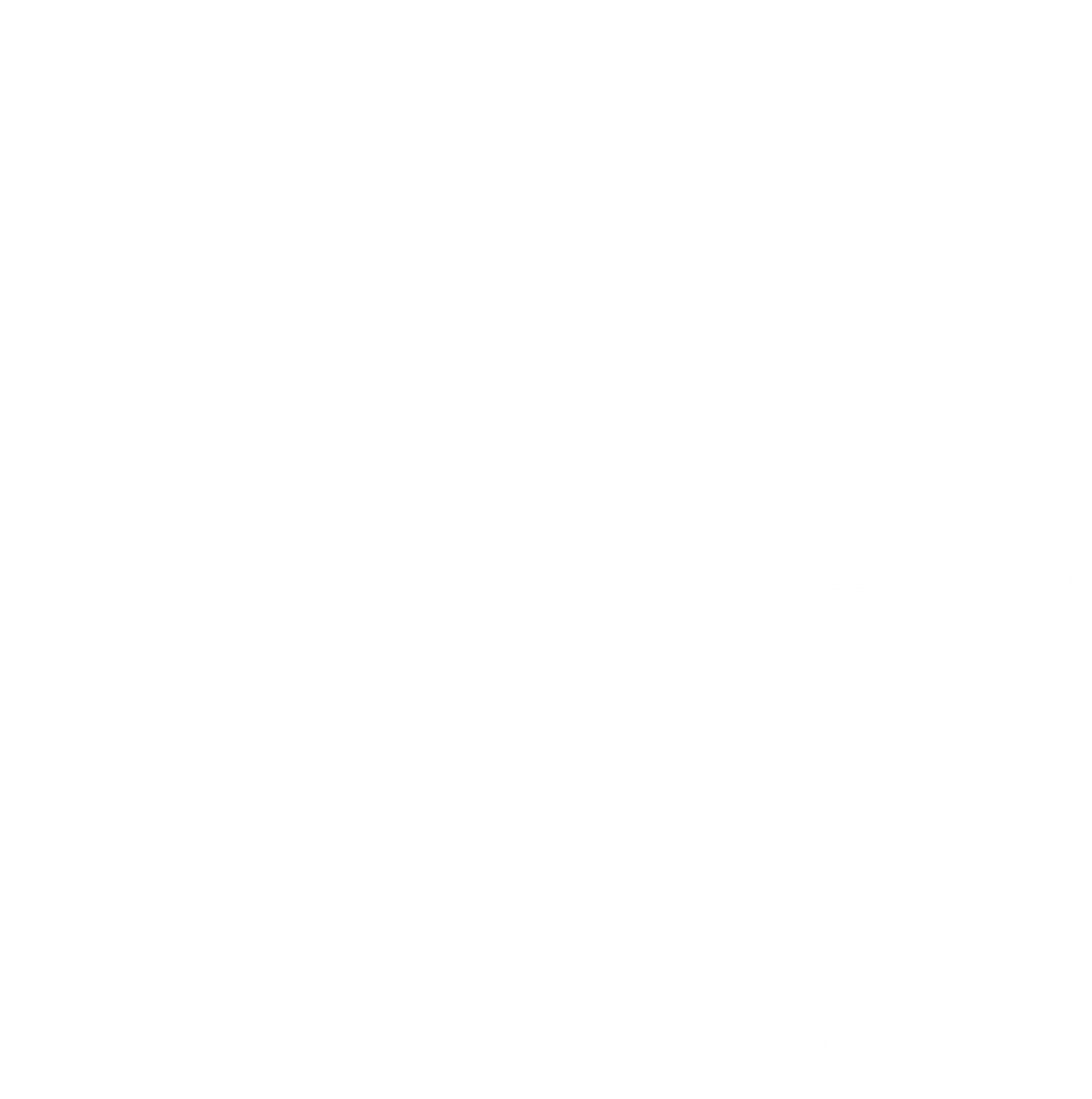
The process of generating output variables is a **stochastic process**, meaning random values are generated over time. For each of the output variables, , , , , we can have corresponding **CDFs**, , , , . Notice how each of the CDFs are depending on the condition . is the **initial condition** of the system.

For an SSQS, the state variables are the length of the queue and the status of the server. For the initial conditions of this system, the server is assumed to be idle and the queue is assumed to be empty.

For an SSQS, say we have an average delay of . The customers that arrive towards the beginning of the simulation will have a far lower delay than this. As more customers arrive, the average delay will increase until it eventually reaches , where it will flatten out. Once the average delay has become **steady**, i.e. it is no longer increasing, the system is said to have reached a **steady state**. Before that time, the system is said to be in a **transient state**.

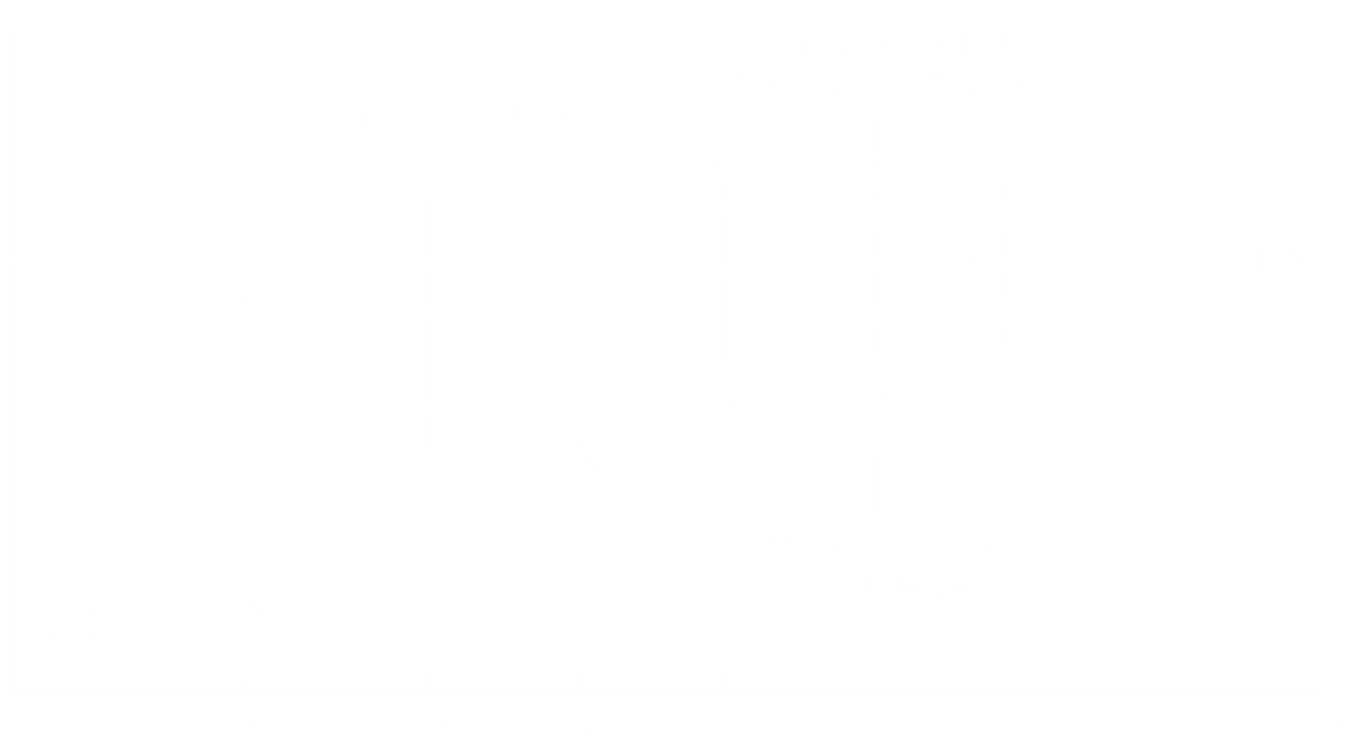
In the transient state, the performance of the system depends on the **initial conditions** and does not represent the **true behaviour** of the system. Thus, the results of the output should be calculated based on the **steady state behaviour** of the system.

For the example of SSQS, if we assume that the initial conditions are that the queue has a certain length, the average queue length, then the transient state can be avoided altogether. Thus, by **controlling the initial conditions**, we can get better outputs. If we consider the graph below, where is the initial queue length, we can see that the initial curve’s variation from the average value (), varies depending on .



Even if we do have a transient period and are not ignoring it, we need to at least ensure that the system is in the steady state for **much longer** than it is in the transient state. In that way, the effect of the transient state will be nullified.

Once the steady state has been reached, the distribution of the random variables becomes **normal**, but before that, it is not.



## Types of Simulations

In terms of output analysis, a simulation program can be of two types:

1. **Terminating Simulations** – Specific conditions have been defined under which the simulation will terminate. For example, the SSQS simulation will stop after customers or time.
2. **Non-Terminating Simulations** – These are simulation where the termination conditions are not defined. We are studying the long-term behaviour of the system, i.e. the steady state behaviour.

Based on the definitions above, it should be clear that it is much more important for **terminating simulations** to identify the **transient period**.

## Statistical Analysis for Terminating Simulations

**Non-Terminating simulations** can have one of two types of parameters:

1. Steady-State Parameters
2. Steady-State Cycle Parameters

A **Steady-State Parameter** applies to a situation where the simulation is continuous. Once the steady state has been reached, the system remains there.

On the other hand, we might have situations where there are breaks in the system. For example, consider that a system consists of a machine that produces some products and some human operators who check the quality of the products. The human operators will have breaks in between their work, which will cause the production rate to fluctuate throughout the day. This results in the system not having a steady state. Instead, we have an average output per day. This average output per day will deal with the **Steady-State Cycle Parameter**.

### Estimating Parameters

We need to estimate these parameters, and to do that, we will be using some statistical tools. We will be running the simulation times, and on each run, we will get outputs. For each of the runs, we can find the **average** value of the outputs, also called the **sample mean**. Thus, for the th run, the sample mean is given by

And from here, the **average output** value, also called the **Point Estimation** of the simulation parameter, is given by

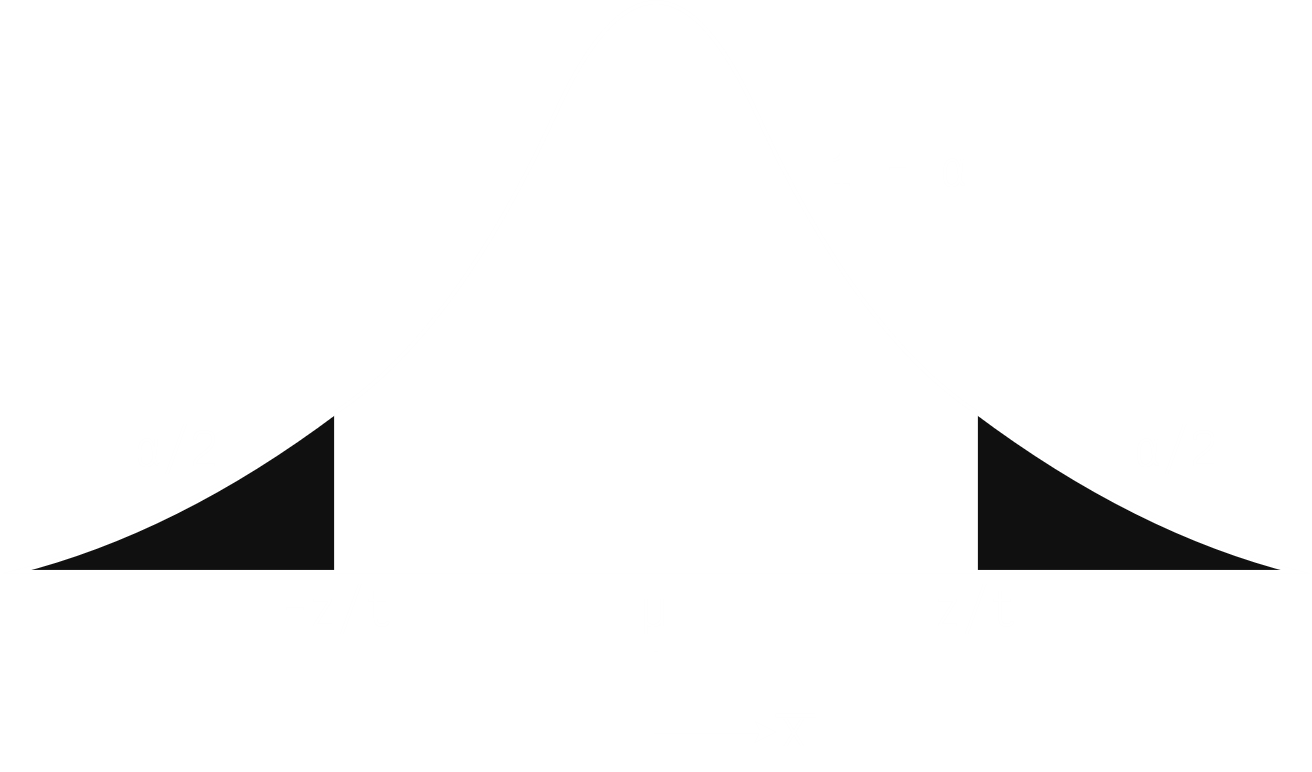
We can also calculate the **variance** as

### Confidence Intervals

We can also measure the **confidence interval**, which is the level of confidence with which we can say that the estimated value of the parameter is within a certain interval.

According to CLT, the distribution of the parameter will be **normal** for sets that are larger than 30 runs.

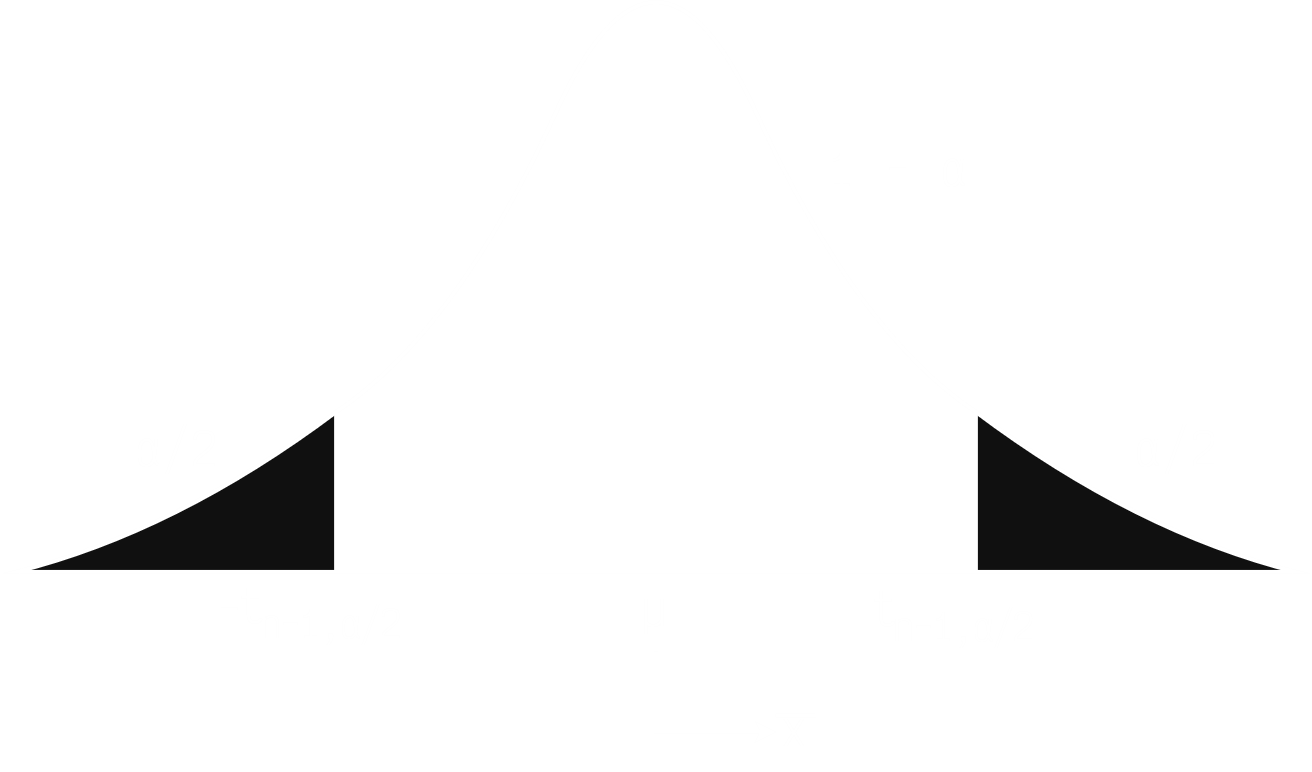
Here, is the **true mean** and for our purposes, is just .



Since this is a normal distribution, we take the curve for the distribution. This curve has a lower and an upper **critical point**, the values of which depend on a variable, . We can say with a confidence that the mean value is within this range.

Notice that in the graph above, the critical points are denoted as . This is because the above distribution could either be a **normal distribution** or a **student t distribution**, depending on the state of . If is **known**, the distribution will be **normal**. It if is **unknown**, the distribution will be **student t** with degrees of freedom. This confusion does not come into play however, if the output values we were getting were normally distributed themselves.

If the distribution is normal, then the range of possible values is . If the distribution is a student t distribution, the range of possible values is .



Note that, for the diagram above, if we want say a confidence interval, the critical values would be at , since the area to the right of the right-most critical point would be .

## Statistical Analysis for Steady-State Parameters

### Welch Procedure

As discussed before, we are forced to **delete some values** from the initial outputs of the simulation since they do not represent the steady-state outputs accurately. This involves 4 steps:

1. Measure the outputs for  **replications** of the simulation program, each providing  **outputs**. Let be the th observation in the th replication. It is assumed that , but is generally a well-accepted value. Additionally, should be much larger than the number of initial outputs that will be deleted, four times as large at least.
2. Measure , the **average of the th observations** from each replication. These values will still exhibit the **transient behaviour**.
3. We will also define a **window** with size , and for this window, we will calculate for

This is called the **moving average**. Essentially, for each value of , the window will define  **values to the left** and  **values to the right**. However, this presents a problem since the values towards the beginning and the end will not have more values on either side. Thus, depending on , we will choose such that there are an **equal number** of values on each side of the th value, up to the maximum possible value for .

Thus, if , then can be at most, this gives us:

1. A **graph** for the different values of is drawn. The **transient behaviour** is said to **stop** at the point where the values **converge**.

Once we have identified the range beyond which the transient behaviour does not exist, we can calculate the **confidence interval** from the values in that range.

In reality, we may have to consider **different window sizes**, and use the graph which is **smoothest**.

**Chapter 10: Comparing Alternate System Configurations**

So far, the output data we have analysed has been for a single system, i.e. we ran a simulation, got some output data, and analysed just that data to get some report, e.g. the average system delay. The goal of a simulation however, is generally to **compare** systems.

We might have an old system which we are trying to replace, so we need to compare the output statistics for the old and the new systems. We could also have situations where we have the exact same system, just the **system configurations** are different, i.e. just the parameters are different. In either case, when talking about just the output statistics, the goal remains the same, to compare the output data.

Consider that we are comparing the **average queueing delay** of two systems, both of which are SSQS. Perhaps in one system we are using a FIFO queue, while in another we are using a shortest-job-first queue. What exactly is different is not relevant right now.

We will essentially be finding the **difference** between the mean values. Once we find that, we also need to perform **hypothesis testing** to ensure that the results we are getting are accurate.

Say we have a series of values, , where is the **system number** and is the **run number**. Thus, we have the average values for **multiple runs** in the two systems.

## Paired Confidence Interval

The first testing method involves subtracting each **pair of runs**, i.e. for each value of . This gives us a new set of random variables, say . For now, we are assuming that both systems have the **same number of data**. If this is not the case, then we **discard some extra data**.

From these, we calculate the **average** value, , and then the **sample variance**.

From this, the **confidence interval** is

## Modified Two-Sample Confidence Interval

Unlike the previous method, where we subtracted the values and used just **one variable**, we will now be using the **two variables** directly. This allows us to bypass the restriction we had, where the two systems had to have an **equal number of runs**.

Once we have these values for both and , we can find the **confidence interval**.

Here, is the **estimated degrees of freedom**.

## Steady-State Performance

To measure the **steady-state performance**, we simply go over the same method, disregarding the values that are a part of the **transient** period. This has been thoroughly discussed earlier and does not merit separate discussion here.