**Chapter 11: Variance-Reduction Techniques**

Table of Contents

[Common Random Numbers 3](#_Toc87178381)

[Configuration Differences 5](#_Toc87178382)

[11.3 Antithetic Variates 9](#_Toc87178383)

The simulation **outputs** we produce are **functions** of the **inputs**. The inputs in turn are **randomly generated**. This means that the **expected values** we get for the output data, i.e. the **average** values, have some **variance** associated with them.

For a particular simulation run, we might not get exactly the expected value, but rather value that is within the range of the variance. The higher the variance, the higher the fluctuation from the expected value. Thus, we need to try to **reduce the variance**.

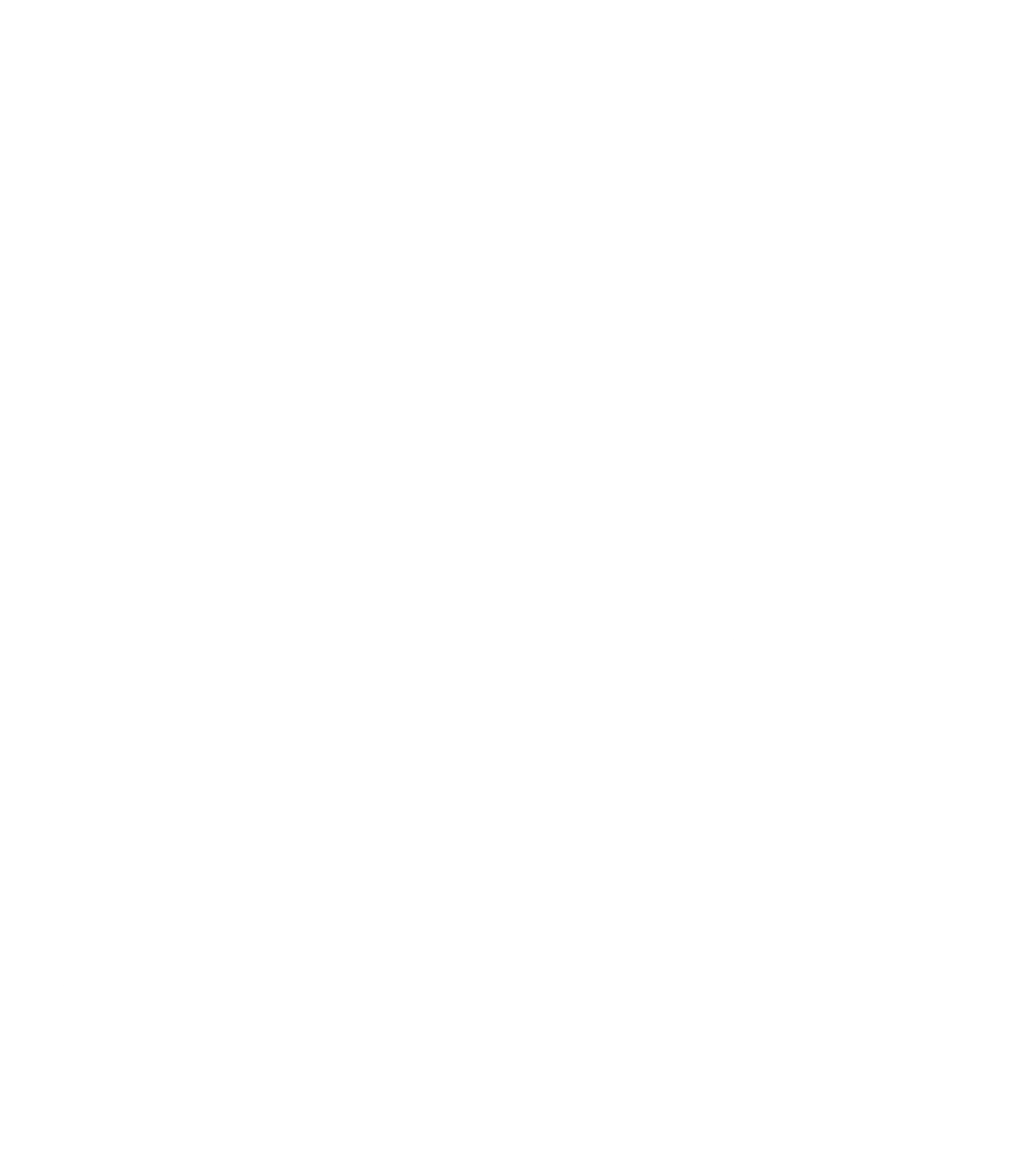
A lower variance also results in a **higher confidence interval**. A low value of will give us a less wide graph, which in turn means that a larger proportion of the values are inside the confidence interval.

We previously studied one technique to increase the confidence interval without changing the value of , which was to simply take **more inputs**. Essentially, of inputs will give us a smaller confidence interval than if we take of inputs. However, taking more inputs is problematic. The variance reduction techniques we will study here will allow us to increase the confidence interval without taking a large number of inputs.

## Common Random Numbers

The **Common Random Numbers** (CRN) mechanism essentially tells us to use the same random numbers in different system configurations. Suppose we are comparing two different systems to determine which has better overall performance. As always, we would be using random numbers for input. If we can ensure that the **same random numbers** are used for both sets of inputs, then the variance when comparing the data will be reduced.

Suppose we have **two configurations** of a system. In the first configuration, we have a **single server** and a **single queue**, while in the second configuration, we have **two servers** and a **single queue**. For the first configuration, the server has an average **service rate** that is **exponentially distributed** with a mean of . In the second configuration, the servers each have a **service rate** that is **exponentially distributed** with a mean of . In both cases, the **arrival rate** is **exponentially distributed** with a mean of . Thus, over time, the two configurations should be able to serve the **same number** of customers within the same time.



We want to compare the **average queueing delay** for both configurations. Let be the average queueing delay for the th configuration in the th run. The **difference** between these values would be . Thus,

From here, if we calculate the **variance**,

Here, is the **covariance** of the two values and is given by

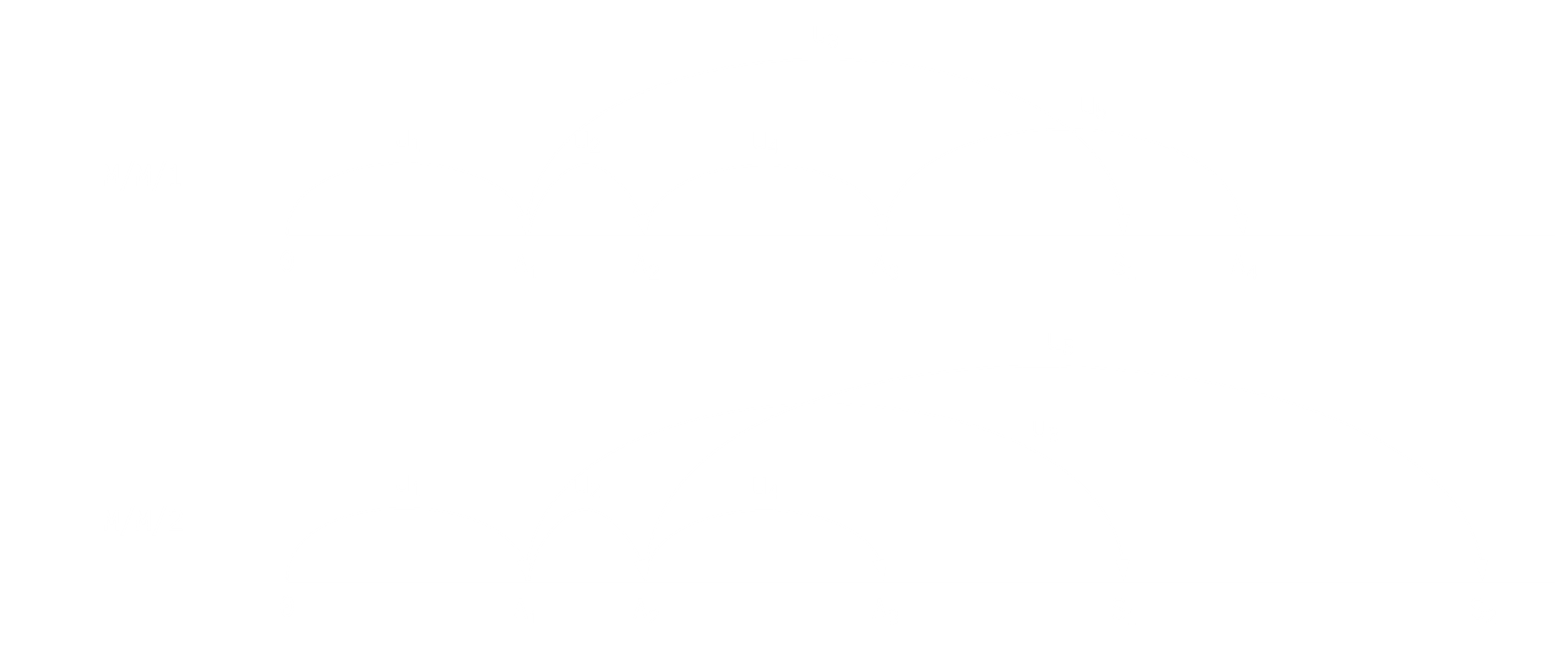
From the above equation, we can tell that is and are **positively correlated**, then the **variance** will be least. We can make them positively correlated by providing **identical** and **synchronized** input. What this means is that not only should the input be the same, but the **order** of the input should be the same as well. Since we are using **random numbers**, we can do this by simply providing the **same seed values** for the corresponding runs of the configurations.

### Configuration Differences

However, there is still an issue. Consider that we have a **single run** of both systems, using the same seed value and thus the same set of random numbers. Say these random numbers are , , . Let’s have a look at the order of events.

First, an **arrival** must be scheduled. Both configurations use to set the arrival times. When the arrival occurs, immediately, the **second arrival** must be scheduled. Both configurations use to do this. In both configurations, there is an **idle server**, so is used to find the **service time** for the first customer.

When the **second arrival** occurs, both configurations use to schedule the **third arrival**. Now however, a difference occurs. In the **first configuration**, the server is **busy**, so a service time is not calculated yet. The customer joins the queue. In the **second configuration** though, the **second server** is **idle**. This means that the second configuration uses to calculate the **service time** of the **second customer**, whereas the **first configuration** will go on to use the same value to calculate the **arrival time** of the **fourth customer**.

****

Even though we used the exact **same random numbers**, due to the difference in **configurations**, we became **unsynchronized**. The values we will get will not be **correlated**.

One way for us to solve this is to use **different streams** for the different input types. Essentially, the arrival times will use one set of random numbers while the service times use a different one. This will prevent us from becoming unsynchronized. Note however, that the different streams all come from the **same cycle** of random numbers. This will become clearer in the example coming next.

Another way to solve this is to calculate the **service time** for each customer when they **arrive**. When they actually get service, that service time will be used. This will also work.

Example 11.2

This example considers an **Inventory System**, with a **minimum order amount**, and **maximum inventory size**, . Such a system has **three input parameters**:

1. Inter-demand Times
2. Demand Amount
3. Delivery Lag

Since we want to compare two systems, for the **first system**, we will consider that and and for the **second system** we will consider that and .

Since we have three input parameters, the **cycle of random numbers** will be divided into **three streams**. Say we have a cycle from to . Thus, the **inter-demand times** will use the stream from to , the **demand amounts** will use the stream from to and the **delivery lags** will use the stream from to .

Since the inter-demand times and demand amounts for both systems will be the same, but the **inventory sizes** are different, the system with the **smaller inventory** will have to order from the supplier **more often**. On the other hand, the **order amount** will be much **smaller**.

There is one problem here. Say we run the systems **multiple times**, and each time, the random numbers used continue from the point in the respective streams where it ended. This will not be a problem for the first two streams, but for the last stream, **delivery lags**, it will.

Say in the **first run** we use  **delivery lag values** for the **first system** but just for the **second system**. In this case, in the **second run**, the first delivery lag value for the **first system** will be the **st value** in the stream, whereas for the **second system**, it will be the **th**. We have become **unsynchronized**.

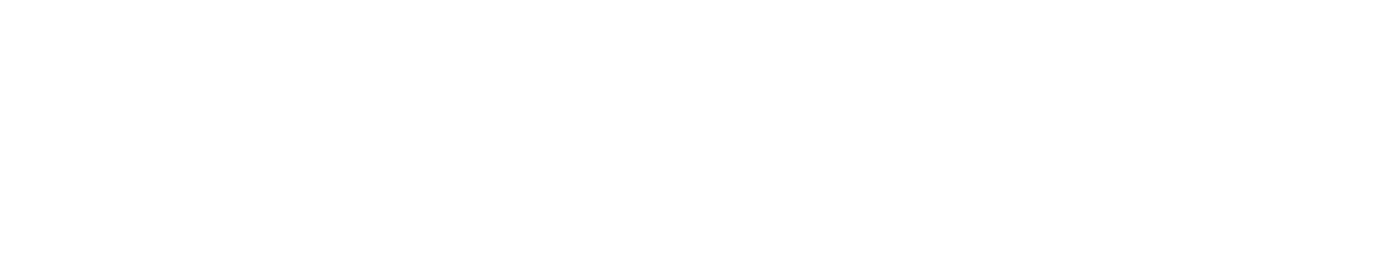
One way to solve this is to use **separate streams** for **each run**, i.e. the second run will use the fourth stream for delivery lag values, the third run will use the fifth stream and so on.

The above solution solves the problem, but there is another one. Within a **single run**, if we use **different random numbers** for **different deliveries**, we will end up with **different delivery lags**. This means that we want both systems to have the **same delivery lags** if they are ordering in the **same month**. This will not be the case in our current set up, since the first system could be using the 10th random number, while the second system uses the 2nd.

The solution here could be to use up a random number whenever we make an **inspection**. Since inspections occur at the same frequency for both systems, the same number of random numbers will be **consumed**, even though not all of the random numbers will be **used**. Thus, in a particular month, if both systems place an order, they will be using the **same random number**. This also simultaneously solves the problem we pointed out first, since **further runs** will also start from the **same point** for both systems.

Example 11.3

This example repeats the example we already say, with an M/M/1 and M/M/2 system, for **four models**. In the **first model** (), we consider that there is **no synchronization**. In the **second** (), we consider that the **arrival times** are synchronized but the service times are not. In the **third** (), we consider the opposite. In the **fourth** (), we consider that **both are synchronized**.



As can be seen in the results above, the **variance and confidence interval** are **least** when both inputs are **synchronized**. Similarly, as we already know, the **highest correlation** is achieved when both inputs are synchronized.

Now to see how this is useful. The variable tells us the **error rate**. Our results should conclude that the M/M/1 queue has a **higher queueing delay**. However, there are going to be runs in which the M/M/1 queue actually has a **lower queueing delay**. This is because the **random numbers** generated, not being **identical**, just so happen to give the M/M/1 system **later arrival times** or **quicker service times**. All this just means that the results we got are due to the **input** and not the system itself. These are **erroneous results**. As can be seen, reducing the variance gives us the **lowest** possible error rate.

## 11.3 Antithetic Variates

We will now consider a method to reduce the variance in a **single model** instead of when comparing with a second model. In this method, we make **pairs** of runs of the model, e.g. the first run and second run are paired, the third and fourth are paired and so on.

Unlike the previous method, in this method, we want the **inputs** of the runs in a pair to be **negatively correlated**. Then, if we use the **average** of the pair instead of the individual runs, small values from one run will be offset by large values in the other run, thus reducing the overall **variance**. More formally, since the values will be **negatively correlated**, the **covariance** will be negative, which will reduce the **variance**, since the equation below will be used to calculate the variance:

Now the question is, how do we ensure that the inputs for the runs in a pair are **negatively correlated**. If we use the **random number** for particular purpose, e.g. to generate the th service time, in the **first run**, we will use the **complementary random number**, , for the **same purpose** in the **second run**. This results in a **negative correlation**, since if gives us a small service time in one run, will give us a large service time in the other run, thus balancing them out in the overall pair.