Master Thesis
Electrical Engineering
July 2014

# ON EFFICIENT AUTOMATED METHODS FOR SIMULATION OUTPUT DATA ANALYSIS

MARTIN BROŽOVIČ

Department of Communication Systems (DIKO)
Blekinge Institute of Technology
37179 Karlskrona
Sweden

This thesis is submitted to the Department of Communication Systems at Blekinge Institute of Technology in partial fulfilment of the requirements for the degree of Master of Science in Electrical Engineering. The thesis is equivalent to 20 weeks of full time studies.


**Contact Information**

Author(s):
Martin Brožovič
Address: Jesenická 25, 794 01, Krnov, Czech Republic
E-mail: martinbrozovic@gmail.com

**External Advisor(s)**

Professor Krzysztof Pawlikowski
Department of Computer Science and Engineering
University of Canterbury
Address: College of Engineering, University of Canterbury, Private Bag 4800, Christchurch 8140, New Zealand
Phone: +64 3 364 2987

**University advisor(s):**

Professor Markus Fiedler
Department of Communication Systems (DIKO)

Department of Communication Systems
Blekinge Institute of Technology
371 79 KARLSKRONA SWEDEN

Internet: www.bth.se
Phone: +46 455 385000
SWEDEN

# Abstract

With the increase in computing power and software engineering in the past years computer based stochastic discrete-event simulations have become very commonly used tool to evaluate performance of various, complex stochastic systems (such as telecommunication networks). It is used if analytical methods are too complex to solve, or cannot be used at all. Stochastic simulation has also become a tool, which is often used instead of experimentation in order to save money and time by the researchers. In this thesis, we focus on the statistical correctness of the final estimated results in the context of steady-state simulations performed for the mean analysis of performance measures of stable stochastic processes. Due to various approximations the final experimental coverage can differ greatly from the assumed theoretical level, where the final confidence intervals cover the theoretical mean at much lower frequency than it was expected by the preset theoretical confidence level.

We present the results of coverage analysis for the methods of dynamic partially-overlapping batch means, spectral analysis and mean squared error optimal dynamic partially-overlapping batch means. The results show that the variants of dynamic partially-overlapping batch means, that we propose as their modification under Akaroa2, perform acceptably well for the queueing processes, but perform very badly for auto-regressive process. We compare the results of modified mean squared error optimal dynamic partially-overlapping batch means method to the spectral analysis and show that the methods perform equally well.

**Keywords:** Akaroa2, batch means, simulation output analysis, sequential coverage analysis, spectral analysis.

# Contents

# List of Figures

# List of Tables

# Introduction

# Chapter 1

# Introduction

With the increase in computing power and software engineering in the past years computer based stochastic discrete-event simulations have become very commonly used tool to evaluate performance of various, complex stochastic systems (such as telecommunication networks). It is used if analytical methods cannot be used. Stochastic simulation has also become a tool, which is often used instead of experimentation in order to save money and time by the researchers. Unfortunately, as shown in [21] stochastic simulations are often used incorrectly, without proper analysis of the output, and then the simulation results cannot be considered credible. In the case of steady-state simulations the problem is that the simulation output data are usually strongly correlated. This has led to proposal of methods of simulation output analysis with various interval estimators. In this thesis, we focus on the statistical correctness of the final estimated results in the context of steady-state simulations performed for the mean analysis of performance measures of stable stochastic processes.

Sequential analysis of output data in steady-state stochastic simulations, used in order to produce final estimates, is nowadays regarded as the approach to use in order to properly control the simulation length and produce appropriately credible final estimates [13], [18]. The simulation runs from checkpoint to checkpoint until a stopping condition is met. In our case we use the relative precision of the final estimate, defined as the ratio of current half-width of confidence interval to the current point estimate of wanted estimation of a performance measure, steady-state mean in our case [17], [20]. The simulation is stopped when such stopping condition is met. One of the main indicators that a method is good is its coverage defined as relative frequency with which the final confidence interval contains the true value $\mu_x$. Any method used for analysis of simulation output data should produce narrow and stable confidence intervals and the experimental coverage should not differ too much from the assumed theoretical level $1-\alpha$.

Pawlikowski argues in [22] that such analysis should be done sequentially in order to produce statistically correct results of the experimental coverage. Pawlikowski outlines rules that should be used for such analysis in [22]. Using these rules we study the coverage of 4 different methods of the mean value analysis, namely Dynamic Partially-Overlapping Batch Means (DPBM) [27], Mean Squared Error Optimal DPBM (MSE-DPBM) [28], modified version of MSE-DPBM (Mod. MSE-DPBM) and Spectral Analysis (SA/HW) as implemented in [16].

Initial transient period is present during the initialization of stochastic processes, it is a period, where the processes do not characterize the steady-state. It has been shown in [17] that a method of detecting the initial transient and truncating all the observations from such period reduces the risk that simulation might stop too early. In [20] it is shown that discarding observation from the initial period leads to reduced bias of the final steady-state estimates. Two techniques are used namely Schruben's test [20] and method of Cumulative Means [6]. The results of the experimental coverage, using variations of the 4 methods of mean value analysis for steady-state systems and one simulation engine per each, have been obtained using a fully automated simulation controller of distributed stochastic simulation Akaroa2 [18].

## 1.1 Research Questions

1. Are DPBM and MSE-DPBM implementable as a tool for steady-state simulation under Akaroa2?
2. Can MSE-DPBM be improved as a method of simulation output analysis?
3. Which of the variants of DPBM perform the best in terms of coverage analysis?
4. Are the DPBM variants accurate as an automated data analysis method in steady-state simulations?
5. Does a variant of DPBM perform better than Spectral Analysis in terms of coverage analysis?
6. Is Schruben's test better than Cumulative Means as a method of initial transient detection?

## 1.2 Aims and Objectives

- To implement DPBM and MSE-DPBM as a component of Akaroa2.
- To modify MSE-DPBM and implement as a component of Akaroa2.
- To implement sequential coverage analysis experiment.
- To compare variants of DPBM and SA/HW in terms of their quality of coverage of confidence intervals using stochastic, analytically tractable reference models.
- To decide upon overall quality of variants of DPBM and SA/HW.
- To decide if Schruben's test performs better than Cumulative Means as a method of initial transient period detection.

## 1.3 Thesis Structure

In Chapter 1 we give an introduction to the research to present our motivation and aims behind the thesis. Chapter 2 describes the background of this thesis. This knowledge helps the reader to understand the fundamental concepts behind steady-state simulations and problems encountered in. Chapter 3 gives an introduction to the simulation output analysis methods (SOAMs). The chapter explains the fundamental theory behind the used variants of DPBM and SA/HW. Chapter 4 presents the initial transient detection methods that are used in the experiment. Chapter 5 presents the analysis of the coverage and rules that have been set up. Chapter 6 describes the necessary changes to implementation of DPBM variants and their implementation as a component of Akaroa2. Chapter 7 provides the results of coverage analysis. Finally, the conclusions are presented in Chapter 8. Future work and discussion are presented in Chapter 8, as well.

# Chapter 2

# Background

This peer review will examine the main issues surrounding the credibility of results while using quantitative sequential stochastic discrete-event simulations as a tool to evaluate behaviour of real world systems. The main focus is given to proper analysis of the simulation output and methods used for such analysis.

In the Sections 2.1 and 2.2 the introduction to quantitative sequential stochastic simulations will be shown. Section 2.3 will include an explanation of valid simulation studies, and how to ensure credibility of the results of simulation studies of various complex stochastic systems. Next in Section 2.4, the sequential and fixed sample size approach to simulation will be shown, the benefits and pitfalls of these two approaches will be discussed. Section 2.5 will introduce the automated controller of distributed simulations Akaroa2. Lastly, the Section 2.6 will describe nowadays vastly used pseudo-random number generators. Various methods of simulation output analysis will be introduced and discussed in Chapter 3, need for such methods is justified in the Section 2.3. Used methods of initial transient detection and their justification will be presented in Chapter 4. Coverage analysis will be introduced in Chapter 5 and its usage, as a tool to assess quality of such output analysis methods, will be justified. Together with the coverage analysis, the reference stochastic processes used for the empirical evaluation of the coverage will be introduced.

At the end of these three major chapters it is hoped that a critical understanding of the key issues is presented, that the reader will be better informed in these areas, and that the research will be justified.

## 2.1 Simulations

Simulations are used to imitate the behaviour and operation of a real-world system, where system is a "collection of entities e.g., people or machines, that act and interact together toward the accomplishment of some logical end [24]". A system state is a collection of variables, which are necessary to describe a system at a particular time. We will focus our attention to *discrete* systems, where the "state variables of a system changes instantaneously at separated points of time [13]", such system can be a stochastic process such as telecommunication network, which we focus on in this thesis. Compared to *continuous* systems, where the state variable change continuously with time. These systems have to be modelled accordingly, one would produce a set of mathematical and logical relationships, called assumptions, about the behaviour and working principle of the system under study. These assumptions would then constitute a model of the system [13].

## 2.2 Computer-based Quantitative Stochastic Simulations

The assumptions, if simple enough, can be solved analytically by using theorems from algebra, probability theory or calculus and provide insight about the performance, behaviour, of the system under various circumstances and input parameters. However, if these assumptions are highly complex, as most of the real-world systems are, the analytical solution would not be possible, either too complex to solve in timely manner or too costly. To overcome this issue computer-based quantitative stochastic simulations are used. In these simulations we use a numerical evaluation of the assumptions of a system under study. The simulation gathers data and based on these data it estimates the wanted performance parameter. Since stochastic processes are solely controlled by random numbers, "the results produced are nothing more than statistical samples [20]". As mentioned in [20] by Pawlikowski, the simulation studies are sometimes taken as a complex programming exercise only and little or no effort is put into proper statistical analysis of the simulation output.

We are focusing on steady-state simulations of stochastic systems, where the simulated processes approach steady-state and the distribution of collected data, observations, become time invariant. Processes are not necessarily in their steady-state as the simulation begins, the steady-state parameters still vary with time. The use of a method to detect this period and truncate such observations can be included to reduce the bias in the final estimates. These methods of initial transient detection are introduced in Chapter 4.

## 2.3 Credibility of results

What is a valid simulation? That was a question that Pawlikowski et al. have asked in [21]. The article has pointed out that one cannot rely on majority of results from research papers published on studies of stochastic system. These studies have been using a simulation as a tool for producing results and deciding upon final claims.

To ensure credibility of a simulation study, experiment, one would need to abide by three basic rules [21]:

1. Use a valid and verified model of a system.
2. Use a correct and tested pseudo-random number generator.
3. Use a proper method of simulation output analysis.

Since the main purpose of this thesis study was solely focused on assessing quality of various output analysis methods, it will be given the most space in this chapter. Section 2.6 will introduce basic introduction and explanation of PRNGs.

Focus is given solely on simulations of stochastic processes in their steady-state, and therefore looking for an unknown mean $\mu_x$ of a wanted performance parameter. This is achieved by taking an average i.e.:

$$\hat{\theta}(n) = \frac{1}{n}\sum_{i=1}^{n} x_i, \tag{2.1}$$

where $x_i$ is the $i$-th observation collected during the simulation, for $i = 1, 2, ..., n$ and is called a point estimator and characterizes the system analysed. To ensure a proper analysis of the output the final estimates $\hat{\theta}(n)$ have to be determined together with their statistical error [22]. The precision with which the point estimator in Equation 2.1 estimates the unknown mean $\mu_x$ is given by:

$$P(\hat{\theta} - \Delta_{1-\alpha}(n) \leq \mu_x \leq \hat{\theta} + \Delta_{1-\alpha}(n)) = 1 - \alpha, \tag{2.2}$$

where $\Delta_{1-\alpha}(n)$ is a half-width of a confidence interval (CI) at a given confidence level $1 - \alpha$ of the point estimator. More precisely, if the simulation is run sufficiently many times and the observations $x_n$ are random variables, the interval at Equation 2.2 would contain the true value of mean approximately $1 - \alpha$ times, example can be seen in Figure 2.1. We call this proportion the coverage by the confidence interval. The $\Delta_{1-\alpha}(n)$ is calculated based on the standard deviation of the estimate $\hat{\theta}(n)$ and assumes that the observations $x_1, x_2, ..., x_n$ are independent and normally distributed (IID) variables:

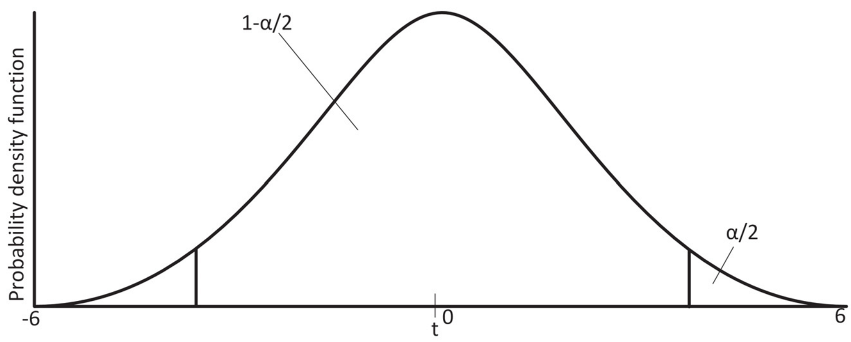


Figure 2.1: The normal distribution of random variable [20]

$$\Delta_{1-\alpha} = t_{df,1-\frac{\alpha}{2}} \hat{\sigma}(\hat{\theta}(n)), \tag{2.3}$$

where $1 - \alpha/2$ is the critical point for the $t$ distribution with $n-1$ degrees of freedom, $\hat{\sigma}(\hat{\theta}(n))$ is the standard deviation of the estimate of the mean [13]. Therefore, the $\sigma^2(\hat{\theta}(n))$, variance of the estimate of the mean, can be called a quality measure, or accuracy, of using the point estimator $\hat{\theta}(n)$. The central limit theorem says, that if $n$ is "sufficiently large", the random variable (estimated mean $\hat{\theta}(n)$) can be assumed to be distributed as a standard normal random variable, regardless of the underlying distribution of the $x_n$ observations [13]. It is well known, that if $n > 30$ the Equation 2.3 becomes [20]:

$$\Delta_{1-\alpha} = z_{1-\frac{\alpha}{2}} \hat{\sigma}(\hat{\theta}(n)), \tag{2.4}$$

where the $z_{1-\frac{\alpha}{2}}$ is the upper critical point obtained from standard normal distribution. For good approximation a value of $n$ should be greater than 100, as recommended in [20] and [13]. Then if the random variables are independent and identically distributed we can use an unbiased estimator of variance [13]:

$$\sigma^2 = \frac{\sum_{i=1}^{n} [x_n - \hat{\theta}(n)]^2}{n-1}, \tag{2.5}$$

However, as it will be shown in Chapter 3, that is not possible in simulation of stochastic processes, where random components are involved. It is due to that the observations are usually highly autocorrelated.

It is also necessary to say that a correct method of detecting the transient period has to be used. Transient period is a period where the processes do not represent steady state yet, these initial observations have to be disregarded to reduce the final bias of the steady-state estimates. Chapter 4 will go into more detail regarding this problem.

## 2.4 Simulation Approach

There are currently two approaches to simulation that are widely used and will be introduced in this section. It is argued that sequential approach has to be used in order to properly simulate different stochastic systems and evaluate the final estimates, of performance measure of systems in the steady-state, together with their statistical errors [18].

### 2.4.1 Fixed sample size approach

Many limitations can be encountered when using this approach. The main issue is that collected sample $n$ of output data can be too small and, because of that, it may not represent steady-state yet or the CI is too large, or in other words: "one does not have control over the size of the CI which results [14]". Additionally, the collected observations are almost always autocorrelated. If these two issues are not properly resolved, the results might differ greatly from the true measure such as steady state mean $\mu_x$ (for example queueing time $\theta_w$ of $M/M/1$ system), since it is impossible to decide in advance how large the sample of output data should be, to make sure that it contains observations representing steady-state behaviour. Generally, results of a simulation run under this approach cannot be relied upon. The main cause of this is that different systems behave in different way and different run lengths are necessary in order to construct adequately small and stable CIs.

### 2.4.2 Sequential approach

"No procedure in which the run length is fixed before the simulation begins can generally be relied upon to produce a confidence interval that covers $\mu_x$ with the desired probability $1-\alpha$, if the fixed run length is too small for the system being simulated [13]". Other justifications for sequential approach can be found here [22], [18]. This method is based on sequential analysis of the quality of the confidence interval i.e.: relative precision of the estimate $\hat{\theta}$, after $n$ observations have been collected, is given as:

$$\epsilon(n) = \frac{\Delta_{1-\alpha(n)}}{\hat{\theta}(n)}, \tag{2.6}$$

where $\Delta_{1-\alpha}(n)$ is the half-width of the CI at the specified $1-\alpha$ confidence level for the estimate $\hat{\theta}(n)$ of the required performance measure $\hat{\theta}$ after $n$ observations [17]. The equation (Equation 2.4) for half-width calculation is presented in Section 2.3.

This approach ,as proposed in [10] by Heildeberger and Welch, takes two arguments: the desired relative precision of CI (as mentioned above), and the maximal run length of the simulation. Both need to be set up before the simulation is started. The sequential simulation uses a sequence of checkpoints. A checkpoint is a point in time at which the $\epsilon(n)$ is compared with the desired level $\epsilon$. If $\epsilon(n) \leq \epsilon$ the simulation is stopped. If $\epsilon(n) > \epsilon$ the simulation proceeds to the next checkpoint, therefore, giving the experimenter a full control over the final error.

## 2.5 Automated Simulation Controller Akaroa2

Akaroa2, developed at the University of Canterbury [18], is an automatic controller based on Multiple Replications in Parallel (MRIP) scenario of sequential distributed simulations. It automatically launches multiple replications, which produce statistically equivalent sequences of observations. These sequences are then provided to the global analyser, which estimates the wanted measure and assesses if the stopping conditions have been met. Akaroa2 implements a modified method of Spectral Analysis (SA/HW) (non-modified version is introduced later in Section 3.2.6) [16] and a sequential version of the classical non-overlapping batch means method (introduced later in Section 3.2.2, as proposed in [20] by Pawlikowski. It has been shown that SA/HW is superior to non-overlapping batch means in terms of quality of the coverage of CIs [22] (it has not been compared to dynamic batch means methods yet, see Section 3.2.5). Akaroa2 is coded in C++ and runs on Unix workstations, it can also be used not only for mean analysis, but also for proportions and quantiles.

Akaroa2 has four main components: *akmaster*, *akslave*, *akrun* and *simulation engines*. Where simulation engines are the processes that run on multiple CPUs, in a LAN, and produce parallel streams of simulation output [5].

### 2.5.1 Akmaster

Akmaster is the global simulation controller, it runs, maintains parallel simulation engines, it provides global output analysis and assesses stopping conditions [5].

### 2.5.2 Akslave

Akslave is the process that runs on hosts, runs simulation engines that are coordinated by akmaster.

### 2.5.3 Akrun

Akrun is the user terminal interface to initiate simulations. It takes several arguments such as simulation name to be run, starting seed, number of simulation engines to use, output analysis method, initial transient method, relative precision and confidence level. Example can be seen in Figure 2.2.

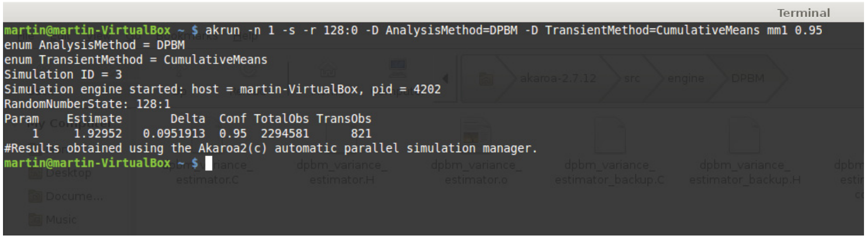


Figure 2.2: Akaroa2 output

When akrun is invoked it contacts the akmaster, which provides an ID of the simulation and chooses a host (running akslave) for each simulation engine based on the requested number $-n$. Akmaster then tells akslave on that particular host to launch a simulation and provides the parameters, as well. Akmaster maintains the host name and port for duplex communication between akslave and itself. Simulation engine generates the data locally on each host and analysis of such data is also done locally by akslave. That means a need for local initial transient detection for each simulation engine. The simulation output analysis is done sequentially, meaning if a checkpoint was reached, akslave sends the local estimates ($\hat{\theta}(n)$ and $\sigma^2(\hat{\theta})$) to the akmaster to be incorporated to the global estimate i.e [5]:

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{N} n_i\hat{\mu}_i, \tag{2.7}$$

$$\hat{\sigma}^2(\hat{\mu}) = \frac{1}{n^2}\sum_{i=1}^{N} n_i^2\hat{\sigma}_i^2(\hat{\mu}_i), \tag{2.8}$$

"where $\mu_i$ is the local estimate from the simulation engine $i$, $\hat{\sigma}_i^2$ is the local estimate of the variance of $\mu_i$, $n_i$ is the number of observations from engine $i$, $N$ is the number of engines and $n = \sum_{i=1}^{N} n_i$ [5]". When the global estimates are calculated the analysis of the stopping condition, such as relative error (Equation 2.6), is performed. If they are reached, akmaster kills the simulation engines and sends the final estimates back to akrun to be presented to the user. The principle of funtion can be seen in Figure 2.3

Akaroa2 uses Combined Multiple Recursive Generator for the generation of random numbers, which will be introduced in Section 2.6. It also provides APIs for the user to implements its own simulation models.

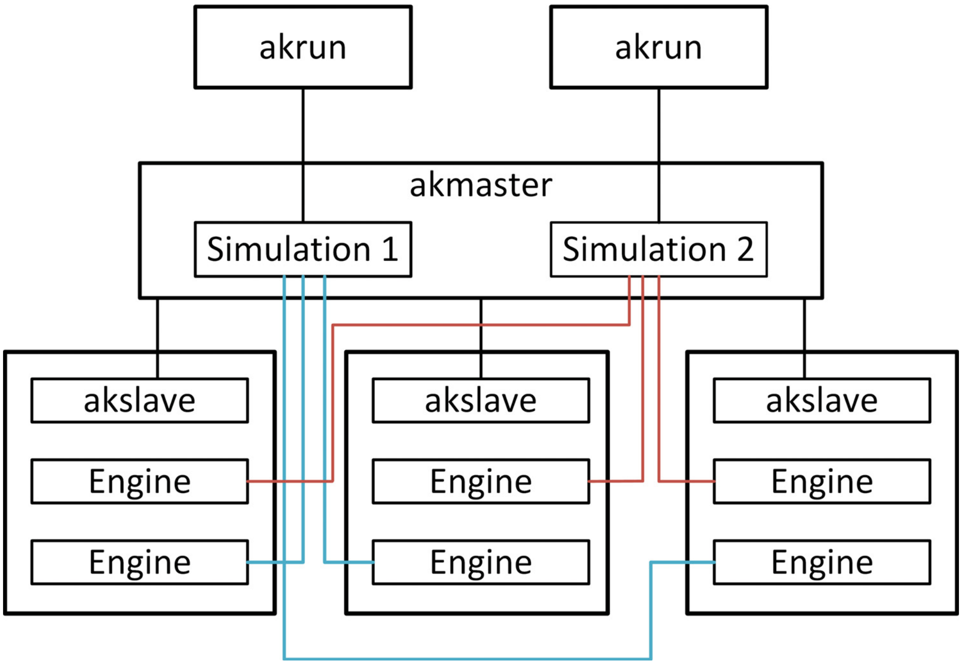


Figure 2.3: Akaroa2 function [5]

### 2.5.4 Distributed Simulations

In this section we introduce the two approaches that are widely used in order to speed up the simulation runs. First, we introduce the Single Replication in Parallel scenario and after we will talk about Multiple Replications in Parallel scenario, a scenario that we use within this research.

#### Single Replication in Parallel

In Single Replication in Parallel (SRIP) scenario is the simulation model split up between a number of simulation engines (processors) or is split into independent sub-models. As mentioned in [23], SRIP does not provide a significant speed up as the level of distributiveness of such model is highly limited, and some models cannot be split up at all. Another problems include that if one replication fails, the whole simulation experiment fails. Therefore, it is not recommended to use such scenario to speed up simulation experiments.

#### Multiple Replications in Parallel

Pawlikowski has mentioned, in [23], that simulation run-length only depends on the time needed to collect the necessary number of observations. This has lead to the idea behind MRIP. In the MRIP scenario independent replications of a simulated model are run on different simulation engines. These engines would then produce statistically different, independent, streams of observations of such simulated model, in our case representing the steady-state mean. The observations are then submitted to one global analyser (Akmaster) which controls the simulation run and decides if such simulation has collected enough observations and the stopping condition has been satisfied, see Section 2.3. If the stopping condition has been satisfied, all the simulation engines are stopped at the same time, even if one engine might be still running a replication.

However, a problem arises here, if one simulation engine is much faster than the other ones a possibility exists that this simulation engine would perform all the work before the slower engines could even finish one replication. On the other hand, MRIP has shown that while used on similar CPUs the speed up is significant and can even improve the quality of used methods of output analysis [22]. In our research we have used MRIP with only one replication (single simulation engine) per model.

For more information please refer to [18].

## 2.6 Pseudo Random Number Generators

Simulating processes that include random components necessarily calls for use of random numbers that are drawn from a specified distribution. Before the incline of simulation experiments, random numbers were drawn by hand, dice rolling, or simple machines. However, they included many problems such as speed of producing numbers or need to save every random number to a memory for later use, such as debugging or reproduction of results. Therefore they cannot be used for computer based stochastic simulations. That lead to use of algorithmic random number generators, where the numbers are produced based on an formula (algorithm). One can see that such numbers are not random at all, but they only appear random, therefore we will talk from now on about pseudo random number generators (PRNGs). Section 7.1 in [13], outlines rules that every proper PRNG has to satisfy. The rules include that PRNG should produce numbers fast, the numbers have to appear distributed uniformly on U[0,1] and cannot be correlated with each other. The generated sequence has to be reproducible, PRNG has be be able to produce independent streams of random numbers, be easily implementable and efficient in time and memory. It also well known that algorithmic PRNGs have a limited period, where if the period passes the random numbers would start to repeat, this leads to another rule that such period should be long enough. Hellekalek gives a good overview of standards for good PRNGs in [11]. As can be seen from the introduced rules, it is not very easy to find a correct generator. Next section will introduce basic PRNGs that have been used widely for computer based stochastic simulations.

### 2.6.1 Linear Congruential Generators

Linear Congruential Generators (LCGs) have been and probably still are one of the most used PRNGs. The sequence of random numbers is is given by [13]:

$$Z_i = (aZ_{i-1} + c)(mod\ m), \tag{2.9}$$

$$u_n = \frac{Z_n}{m}, \tag{2.10}$$

where $m$ is the modulus, $a$ is the multiplier, $c$ is the increment and $Z_0$ is the starting point, or seed. As mentioned above, the sequence of $Z_1, Z_2, ..., Z_n$, is not random at all. So the selection of parameters has to be given a special attention. The integers have to satisfy $0 < m$, $a < m$, $c < m$ and $Z_0 < m$ [13]. It is quite clear that such sequence would eventually exhaust all possible random numbers, amount depends on parameter selection, and loop itself around and produce same random numbers. If parameters are selected properly it is expected that LCG will produce sequence of random numbers from 0 to $m$, called full period. The seed $Z_0$ is the only parameter that has to be remembered in order to reproduce same sequence of random numbers. The random number $u_n$ would then be a independent and identically distributed in the range [0,1) [25].

#### Multiplicative LCG

Probably the most commonly used PRNG based on LCG is the multiplicative LCG (MLCG). The parameter $c$ is here selected to be 0, $c = 0$. But, by carefully selecting the parameters $a$ and $m$, a period of $m - 1$ can be achieved. It has been shown that $m$ can be selected as $2^b$, where $b$ is the number of bits that the computer, or compiler, has available. Therefore on 32-bit computer $m = 2^31$. However, this is not sufficient as it does not guarantee that such generator will produce a full period. In MLCG the parameter $m$ is selected as the largest prime number less than $2^b$ and if $a$ is the primitive element modulo $m$, that is the smallest integer $a^l - 1$, we can obtain a almost full period of $m-1$, where each random number would repeat exactly once in period [13].

### 2.6.2 Combined Multiple Recursive Generator

As it can be seen above MLCG could produce only a relatively small amount of random numbers ($2^b - 1$). That is solved by the multiple recursive generator (MRG) which is given as follows [25]:

$$Z_n = a_1 x_{n-1} + a_2 x_{n-2} + ... + a_k x_{n-k} (\ mod\ m), \tag{2.11}$$

$$u_n = \frac{Z_n}{m}, \tag{2.12}$$

and can achieve up to period of $m^k - 1$. However, as [25] mentioned, one would like to keep coefficients of recurrence 3 small.

Solution to this problem was introduced in [15] by L'Ecuyer et al. They have introduced combined MRG, which combines a J amount of MRGs [25]:

$$Z_{j,n} = a_{j,1} x_{j,n-1} + a_{j,2} x_{j,n-2} + ... + a_{j,k_j} x_{j,n-k_j}\ mod\ m, \tag{2.13}$$

$$u_n = (\sum_{j=1}^{J} \frac{x_{j,n}}{m_j})\ (mod\ 1)\ (j = 1, 2, ..., J), \tag{2.14}$$

We will focus here on a MRG32k3a [15], as it the one implemented in Akaroa2. MCRG32k3a is defined by two MRGs each with three terms as follows [25]:

$a_{11} = 0$

$a_{12} = 1403580$

$a_{13} = -810728$

$m_1 = 2^{32} - 209$

$a_{21} = 527612$

$a_{22} = 0$

$a_{23} = -1370589$

$m_2 = 2^{32} - 22853$

As mentioned in [25] by McNickle, this implementation performs well in statistical tests up to 45 dimensions. The main advantage here is that MRG32k3a achieves a period of $2^{191}$ with arbitrary seed and at least one non-zero element per MRG. The implementation in Akaroa2 takes the full period and splits it into $2^{127}$ streams of IID numbers. Each of those is then split into $2^{51}$ sub-streams of length $2^{76}$.

More information on PRNGs can be found in [25], [13], [11] and CMRG can be found in [15].

# Chapter 3

# Methods of Output Analysis

Due to the random nature of the simulated processes and their often naturally autocorrelated (not independent) features, output data cannot be analysed using classical statistical methods. Secondly, an initial transient period is present, a period where the processes do not characterize steady-state. The initial observations have to be disregarded to reduce the final bias of steady-state estimates. It has also been shown in [17], that initial transient deletion is necessary in order to reduce the risk of simulation stopping prematurely.

Many methods to properly analyse the simulation output and deal with the problems with correlation have been proposed. Three "basic" methods are: independent replications (IRs), non-overlapping batch means (NBM) and spectral analysis (SA). Also we assume that observations $x_1, x_2, ..., x_n$ are from a covariance stationary process, steady-state mean and variance exist, so we can use the following methods.

## 3.1 Independent Replications

Independent replications have been widely used with fixed-sample size approach to simulation. This method sets up the sample size $n$ prior of the run of the simulation. The sequence of $n$ observations $x_1, x_2, ..., x_n$ can be then split between $k$ independent replications, with $m = n/k$ observations in each. From this a problem arises, where every replication requires to detect transient period at the beginning of each of the replications of the process and discard the observations from this period creating a lot of unnecessary overhead. If the initial sample size $n$ is set up being to small then the estimations of mean will be highly autocorrelated resulting in difference between theoretical coverage $1-\alpha$ and the experimental coverage of the CIs. This is due to that if one increases number of $k$ replications it would lead to smaller batch size $m$ per replication, and therefore increasing the size of

CI for $\hat{\theta}$, not the actual value of $\hat{\theta}$ [14].

For further details please refer to in [20], [13], [14].

## 3.2 Batch Means Methods

Methods based on batch means split a single run of length $n$ into $k$ batches of size $m$. Which, if long enough, can be assumed to be independent from each other. The mean is then estimated per batch and contributes to the overall mean over the $n$ observations. The batches can also arbitrary overlap resulting in decrease of the variance of the estimator.

There are methods developed for both fixed and sequential approach of simulation and the most important ones will be introduced in this section.

### 3.2.1 Non-overlapping batch means

Method of non-overlapping batch means (NBM), as proposed originally by Conway [2], is traditionally used with fixed sample size approach. As described above, the sample size is decided before the actual simulation run and the $n$, $x_1, x_2, ..., x_n$, observations are split into $k$ batches of size $m$. The mean is constructed according to the Equation 2.1 and is, therefore, decided from the individual observations and is equivalent to calculating the mean per batch, see Equation 3.2. The variance, or quality measure of the point estimator, is estimated from all of the means over all batches see Equation 2.5, or can be estimated based on the individual batch means, see Equation 3.1.

As mentioned in [20] and [13], the main idea behind this approach is that observations that are separated more in time are less correlated. So if we set up the batch size to be long "enough" the batch means might seem as uncorrelated and normally distributed, which also flows from the Central Limit Theorem as mentioned in Section 2.3. However, this leads to the biggest problem of this method being how to decide if the batches are long "enough". The problem is that if the run length $n$ is too "small" the means over individual batches can be highly correlated and the estimator in Equations 2.5 and 3.1 will be heavily biased. Both of these will negatively influence the size and quality of final CIs, therefore resulting in lower coverage than $1 - \alpha$. Second problem would be how to determine the length of the initial transient period and therefore from which observation can one start creating the batches, because all the observation in batches need to characterize the steady-state of a stochastic process. Transient detection methods will be discussed in the Chapter 4. Idea of batching in NBM is shown in Figure 3.1.

$$\hat{\sigma}^2 = \frac{\sum\limits_{j=1}^{B} (\bar{X}_j - \hat{\theta}(n))^2}{b(b-1)}, \tag{3.1}$$

where

$$\bar{X}_j = \frac{\sum\limits_{i=(j-1)}^{jm} x_n}{m} \tag{3.2}$$

is the $j$-th mean of a batch.

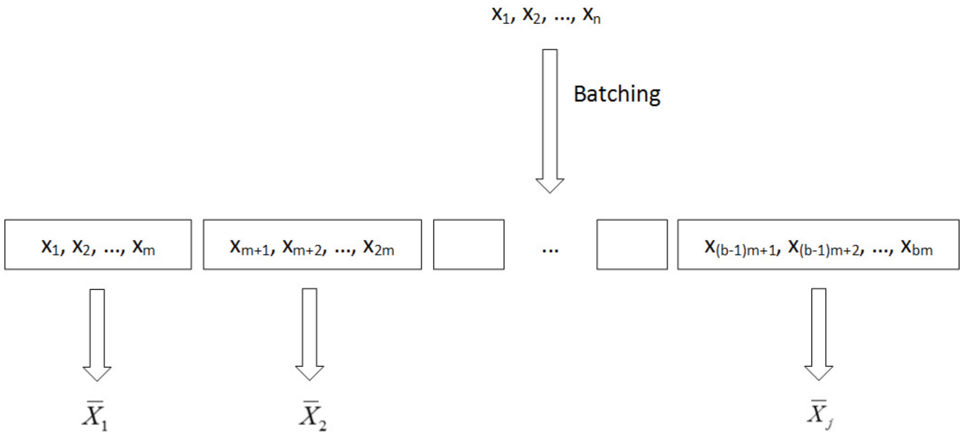


Figure 3.1: NBM Batching

### 3.2.2 Sequential implementation of NBM

The sequential implementation of non-overlapping batch means, as described in [20], will be described in this section. For the sequential implementation of NBM one would need to specify two phases. First, a phase where the initial period of a process is detected and observations from such period are discarded. Second, a phase where the process is simulated and its steady-state performance parameter is analysed. The second phase needs the observations to be representing the steady-state of a process, which is achieved by using the phase one.

As was mentioned above, the observations collected during the simulation are usually highly autocorrelated, therefore the classical statistical methods cannot be used, as they assume independence of data. The main problem with classical NBM is that if the batch size is selected incorrectly, the overall quality of coverage of CI is affected, resulting in usually lower coverage than selected $1 - \alpha$. This is solved by this sequential method, the batch size $m^*$ is selected sequentially here. The implementation in [20], does not keep the observations separately, but rather the means over batches are kept. These batches are then tested for autocorrelation. "A given batch size can be accepted as the batch size for approximately uncorrelated batch means if all $L$ autocorrelation coefficients of lag $k(k = 1, 2, ..., L)$, are statistically negligible at a given significance level $\beta_k; 0 < \beta_k < 1$" [20]. Also it is not necessary to estimate autocorrelations coefficients over all means, it is usually enough to estimate only $0.1k_{bo}$ lags, where $k_{bo}$ is the number of batch means used for autocorrelation test. This is given due to that with increasing lag, the autocorrelation coefficients are calculated from lower amount of data, and therefore negligible. The method uses a estimator referred to as "jackknife" for the autocorrelation estimation. This estimator is usually less biased than the ordinary estimators, i.e. [20]:

$$\hat{\hat{r}}(k, m) = 2\hat{r}(k, m) - \frac{\hat{r}'(k, m) + \hat{r}''(k, m)}{2}, \tag{3.3}$$

where $k$ is the lag coefficient and $m$ is the batch size, "$\hat{r}'(k, m)$ and $\hat{r}''(k, m)$ are estimators over the first and second half of the analysed sequence of $k_{bo}$ batch means" [20]. It is also recommended that $k_{bo} \geq 100$, size of batches $m$ should not be less than 50 and that $L$ should not be too large, just about $0.1k_{b0}$ as mentioned earlier.

The method holds two buffers, the "ReferenceSequence" is used for holding means over all the batches of size $m_o$, and "AnalysedSequence" used to hold $k_{bo}$ number of batches of size $m_s = sm_o, (s = 1, 2, ...)$, which is formed from the batch means kept in the reference sequence. The batch size $m^* = m_s$ if the batch size passed the autocorrelation test two consecutive times.

For more detailed explanation and implementation please refer to [20].

### 3.2.3 Overlapping Batch Means

Overlapping batch means (OBM), as originally proposed by Meketon and Schmeiser in [19], exploit the idea that if you create a new batch with each consecutive observation, you will have more observations in the final estimation of the mean and variance of the mean. However, as it can be seen it also suffers from greater correlation between the means. On the other hand, as was mentioned above, the batch size is more important than the independence between the individual batches. The article in [19] has also shown that the estimator used with OBM has variance 1/3 lower than the "classical" NBM estimator and that it will essentially have 1.5 times greater number of degrees of freedom, while keeping the bias of the estimator the same. The method was developed mainly for fixed sample size approach, where one could have more observations per batch than in the NBM and, therefore, save computational time. Idea of batching in OBM is shown in Figure 3.2.

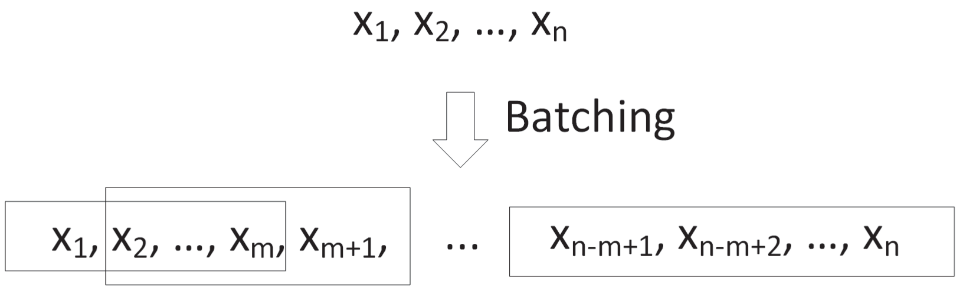


Figure 3.2: OBM Batching

For more information please refer to [19] and [20].

### 3.2.4 Dynamic Non-Overlapping Batch Means

The method, as proposed by Yeh and Schmeiser in [30], does not require the run length to be set before the simulation run. Dynamic non-overlapping batch means (DNBM) creates batches on a same principle as NBM does, and therefore splits the sample of $n$ observations into $k$ smaller, independent, batches of size $m$. DNBM uses finite memory and manipulates the batch size dynamically during the actual simulation run. The methodology here is very similar to the one introduced in [20] for the Spectral Analysis method. The method shows good memory (O(1)) and computational requirements (O($N$)).

The main idea is to create a vector, finite memory space, of size $2k(k = 1, 2, ..., 2k)$ where $k$ is a positive integer, that will hold the observations. This vector constitutes of cells, batches of size $m$. DNBM holds the observations as sums and each new observation $x_n$ is added to the sum. If the current batch (cell of a vector) becomes full the algorithm would move to the next batch, if any, and add the observation until this batch becomes full. When there is no more space in the memory i.e. all the batches are full, equal to the batch size $m$, DNBM would "collapse" the $2k$ batches in vector into $k$ batches. Meaning the means from batches $k + 1$ up to $2k$ will be added to the batches 0 up to $k$ and the batch size would be doubled $m = m \times 2$. Hence, half of the vector would essentially become available. The batch size $m$ can be determined from $n$ and $k$ [30]:

$$m = 2^{\lceil log_2 \frac{n}{k} \rceil - 1}. \tag{3.4}$$

Therefore, the batch size will always increase by the power of two. It is recommended to use between 10 to 30 batches i.e. $k = 5$ to 15 [30]. The method also introduced two estimators, one $\hat{V}_{TBM}$ that would not consider a "partial batch", where partial batch is the last batch that has not been filled up at the current checkpoint, and would truncate all the observations from such batch. Second, $\hat{V}_{PBM}$, that would consider all the batches as they are. It has been shown in [30] that in terms of mean squared error (MSE), MSE will be introduced in Chapter 5, the $\hat{V}_{PBM}$ shows overall lower MSE especially for small number of batches $k$ the small partial batch helps to reduce the variance more significantly than it increases the squared bias. For higher number of batches the difference becomes negligible. However, this experiment was done only for few processes and the experimental quality of coverage was not assessed at all. Both estimators can be seen in following formulas:

$$\hat{V}_{TBM} = \frac{\frac{m}{n}}{b-1} \sum_{i=1}^{b} (\frac{A(i)}{m} - \hat{\theta}_{bm})^2, \tag{3.5}$$

$$\hat{V}_{PBM} = \frac{\sum_{i=1}^{r_A - 1} ((\frac{A(i)}{m} - \hat{\theta}(n)))^2 + (\frac{m_A}{m})((\frac{A(r_A}{m_A}) - \hat{\theta}(n))^2}{(r_A - 1)(r_A - 1 + \frac{m_a}{m})}, \tag{3.6}$$

where $A$ is the vector of size $2k$, $m$ is the current batch size, $n$ is the sample size, $\hat{\theta}(n)$ is the overall sample mean, $\hat{\theta}_{bm}$ is the sample mean of truncated data, $b = \lfloor \frac{n}{m} \rfloor$ is the number of full batches, $m_A$ is the batch size of a current batch pointed to by $r_A$. Idea of batching in DNBM is shown in Figure 3.3.

For detailed description please refer to [30].

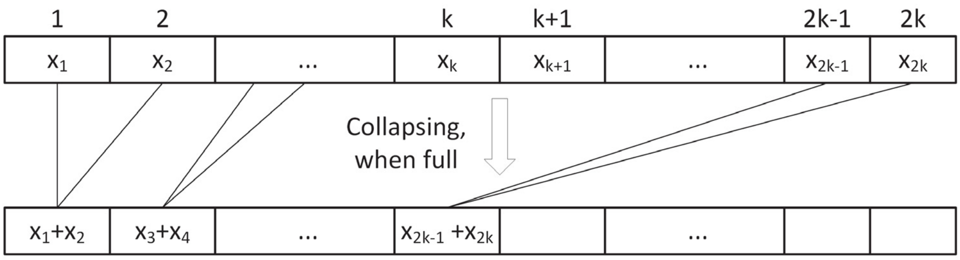


Figure 3.3: Collapsing in DNBM

### 3.2.5 Dynamic Partial-Overlapping Batch Means

Dynamic partial-overlapping batch means (DPBM) were proposed originally in [27], [26] and [28] by Song. The main idea here is to collapse a vector of a certain size in the same way as in DNBM. But to use a special case of OBM called partial overlapping batch means (PBM), as can be found in [29]. The idea behind PBM is to shift the overlap and do not create a new batch with each new observation $x_n$ as in OBM. The shift has been selected to be $m/4$ and it has been shown, see [27], that PBM estimator with $m/4$ shift has only 3 % higher asymptotic relative variance than OBM see Table 3.1 while reducing the correlations between the batch means. To implement this idea in DPBM it is required to have 4 vectors of size $2k$, where all the data will be stored. The idea is to collapse all the vectors when the first vector becomes full. The idea of collapsing the first vector is identical to the one of DNBM, however to create a 75 % overlap the observations have to be collapsed into second, third and fourth vector, as well. When the first vector becomes full for the first time the data are collapsed only into the first and second vector. Every other collapse the data are collapsed into all four of the vectors. After collapsing has occurred for all of the vectors it is necessary to save the observation $x_n$ into all of them. For detailed algorithm see [27]. The idea of collapsing can be seen in Figure 3.4.The variance estimator is described as following [27]:

$$\begin{aligned}\hat{V}_{DPBM} =&\frac{1}{d_b}\Big[\sum_{i=1}^{b_1}\Big(\frac{A_k(i)}{m}-\hat{\theta}(n)\Big)^2\\&+\sum_{i=1}^{b_2}\Big(\frac{B_k(i)}{m}-\hat{\theta}(n)\Big)^2\\&+\sum_{i=1}^{b_3}\Big(\frac{C_k(i)}{m}-\hat{\theta}(n)\Big)^2\\&+\sum_{i=1}^{b_4}\Big(\frac{D_k(i)}{m}-\hat{\theta}(n)\Big)^2\Big],\end{aligned} \tag{3.7}$$

where $b_i, i = 1, 2, 3, 4$ are number of full batches in vector $A$ to $D$ [27]:

$$b_1 = r_1 - 1 + \left\lfloor \frac{m_1}{m} \right\rfloor, b_2 = r_2 - 1 + \left\lfloor \frac{m_2}{m} \right\rfloor, b_3 = r_3 - 1 + \left\lfloor \frac{m_3}{m} \right\rfloor, b_4 = r_4 - 1 + \left\lfloor \frac{m_4}{m} \right\rfloor, \tag{3.8}$$

,$d_b = n((\frac{n}{m}) - 1)$, $b = \left\lfloor \frac{(n-m+s)}{s} \right\rfloor$ and $s = m/4$. See [27] for proof. The batch size amounts to $m = 2^h$, where $h$ is the number of collapses.

However, the DNBM and the original DPBM [27] do not reflect the correlation structure of the data as the batch size is selected only on a basis of $n$ and $k$. To overcome this problem a mse-optimal DPBM (MSE-DPBM) estimator was proposed in [28]. The algorithm of MSE-DPBM would take a value of current estimated variance with batch size $m$, $\hat{(V)}_D PBM(m)$, as

| i | Shift s=m/i | Estimator Type | $\frac{\sigma^2(\hat{V}(m,s))}{\sigma^2(\hat{V}^O(m))}$ | $\frac{bias(\hat{V}(m,s))}{bias(\hat{V}^O(m))}$ |
|---|---|---|---|---|
| 1 | $m$ | NBM | 1.50 | 1 |
| 2 | $m/2$ | PBM | 1.12 | 1 |
| 3 | $m/3$ | PBM | 1.06 | 1 |
| 4 | $m/4$ | PBM | 1.03 | 1 |
| $m$ | 1 | OBM | 1.00 | 1 |

Table 3.1: Asymptotic bias and variance result [27]

a baseline to estimate the optimal batch size $\hat{m}^*$. It would then adjust the value of batch size, or memory size, accordingly to the value of $\hat{m}^*$ to reflect the correlation of the data [28]. If the current batch size $m$ is far greater than $\hat{m}^*$ the algorithm increases the memory $k, k = k + 1$ and no collapsing occurs. The crucial step is therefore to estimate the $\hat{m}^*$ and for sufficiently large $m$ and $n$ it is:

$$\hat{m}^* = (1.03n(\frac{\hat{\gamma}_1}{\hat{\gamma}_2})^2)^{\frac{1}{3}} + 1, \tag{3.9}$$

where $\hat{\gamma}_0$ is the sum off all correlations:

$$\hat{\gamma}_0 = n\hat{V}_{DPBM}(m)/\hat{R}_0, \tag{3.10}$$

$\hat{\gamma}_1$ is the sum of all weighted correlations:

$$\hat{\gamma}_1 = nm[\hat{V}_{DPBM}(m) - \hat{V}_B(\frac{m}{2})]/\hat{R}_0, \tag{3.11}$$

and $\hat{R}_0$ is the sample variance:

$$\hat{R}_0 = n^{-1}(\sum_{i=1}^{n} x_i^2 - n\hat{\theta}^2). \tag{3.12}$$

$\hat{V}_{DPBM}(m)$ is the current estimated variance Equation 3.7 and $\hat{V}_{DPBM}(m/2)$ is the previous estimated variance, or variance that was calculated before the batch size was doubled (collapsing) and equals to 50 % OBM estimator proposed in [26] i.e.:

$$\hat{V}_B(m/2) = \frac{1}{d_b}[\sum_{i=1}^{b'_A}(\frac{A'_{j-1}(i)}{m_{j-1}} - \hat{\theta}_n)^2 + \sum_{i=1}^{b'_B}(\frac{B'_{j-1}(i)}{m_{j-1}} - \hat{\theta}_n)^2], \tag{3.13}$$

$m_{j-1}$ is the previous batch size at previous step $j$, or $m/2$, $d_b = n((\frac{n}{m_{j-1}})-1)$, $b = \left\lfloor \frac{(n-m+s)}{s} \right\rfloor$, and $A'_{j-1}, B'_{j-1}$ are virtual vectors, constructed to represent a previous state of vectors, because the previous state of the four vectors is overwritten at step this step. See Equations (13)-(16) in [26].

More information can be found in [27], [26] and [28].

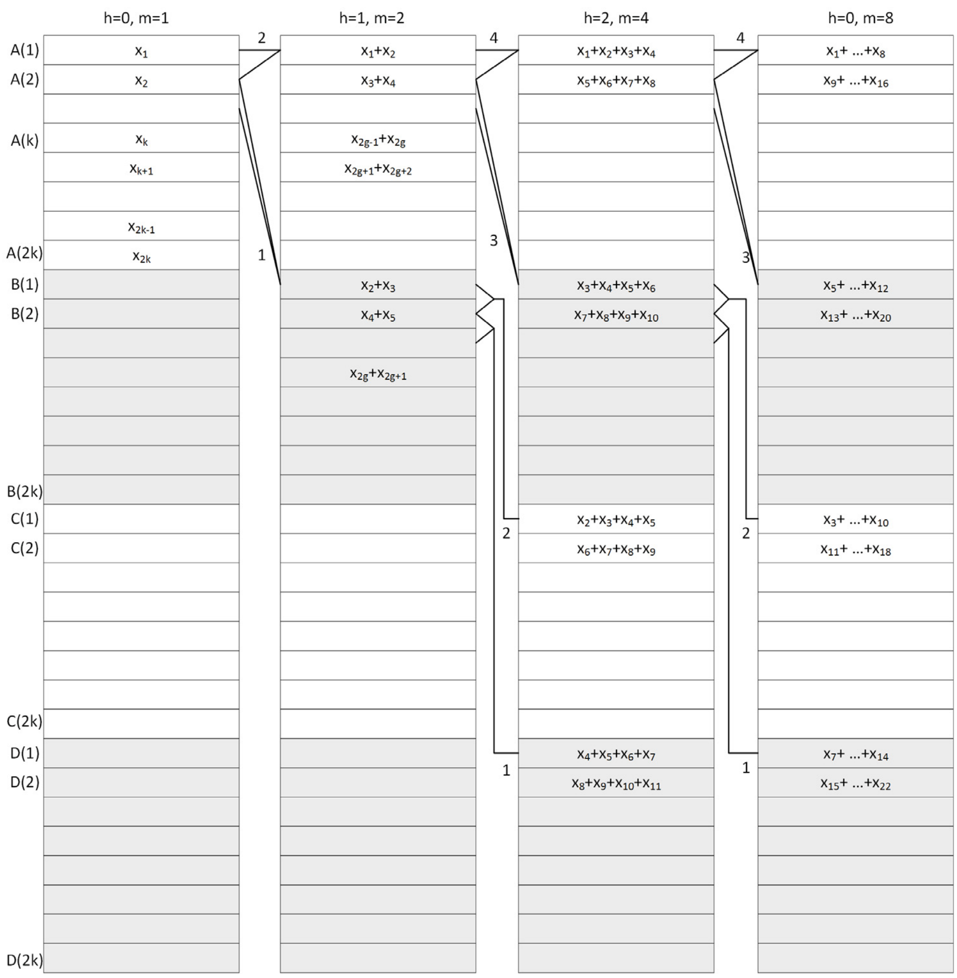


Figure 3.4: DPBM Collapsing [28]

### 3.2.6 Spectral Analysis

Assuming that the process has already passed its initial transient period and the observations characterize the steady-state of a process the Spectral Analysis (SA) can be used.

Spectral Analysis was initially proposed in [9] and modified for use under Akaroa2 [18] and [16]. From a covariance stationary 4.1 sequence of observation $x_1, x_2, ..., x_n$ assume that $\gamma(k)$ is the covariance function given by [9]:

$$\gamma(k) = E[(x(j) - \hat{\theta}(n)(x(j+k) - \gamma], \tag{3.14}$$

and $\sum_{k=-\infty}^{\infty} \|\gamma(k)\| < \infty$. So that the process has a spectral density $p(f)$ and the functions $\gamma(k)$ and $p(0)$ are, therefore, Fourier pairs [9]. It is well known that the variance can be estimated $\hat{\sigma}^2 = p(0)/n$, the spectral density at frequency zero divided by number of observations.

SA does not suffer from the problems with autocorrelated data. Original implementation by Heidelberger and Welch does plan for the method to be used in the sequential approach and also groups the collected observations into finite number of batches to save memory, rather than saving all the observations. It can be seen that the main problem here is how to estimate the $p(0)$. "The variance is obtained as the value of the periodogram $\Pi(f)$ (of the analysed sequence of observations) at the frequency $f = 0$" [3]. As the peridogram, especially for low $f$s, is highly variable the periodogram is transformed into a logarithm of an averaged periodogram to achieve a smoother function. A regression fit to such logarithm of the averaged periodogram is applied to obtain the value of the periodogram at $f(0)$. The polynomial degree of the regression fit is usually less than two, $d \leq 2$ using a fixed number, $K$, of the points of such periodogram $\Pi(f)$. In [9] authors show that to estimate the steady-state mean $\mu_x$ we should use $d = 2$ and $K = 25$ and the CI of such $\mu_x$ can be constructed using quantiles of the Student $t$ distribution with degrees of freedom equal to 7 ($df = 7$) [3]. Various values for $d$ and $k$ have been tested in [8], but no significant change was achieved. For more detailed explanation please refer to [9], [20] and [3].

This method, as mentioned above, has been modified for use in Akaroa2 in [20] and later modified in [16]. In [20] the method takes two arguments, relative precision of the estimates and maximal simulation run length. Sequential approach using checkpoints, as described in Section 2.4.2, is used and selection of checkpoints is explained in detail in [20]. The method uses a finite memory and groups the observations into batches $M$, where the minimal number of batches is 100. A mean $\hat{\theta}(n)$ is then calculated per each of the batches and observations are, therefore, kept as their means to save memory. When $M$ batches with batch size $m$ are collected half of the batches, from middle to the end, are collapsed into the batches in front and doubling the batch size. It has been found in [16] that SA/HW performs very well in terms of quality of the CIs when the acceptable error is greater than 10 %. However, the quality lowers when a higher precision is necessary as the simulation can stop prematurely. The method in [16] swaps the polynomial used for the regression fit to the average of the periodogram points, which would equal to using a polynomial of degree 0.

# Chapter 4

# Initial Transient Detection

The purpose of this sections is to introduce initial transient period detection methods and steady-state processes that have been used within the experiments in this thesis.

## 4.1 Stationarity

As was said in Chapter 3 the methods of output analysis, and is important for our research, assume that the observations collected during the simulation of a stochastic process characterize the steady-state behaviour of a simulated process. And that the processes simulated are stationary [12]:

$$F_X(x; t + \tau) = F_X(x; t), \tag{4.1}$$

have constant mean, constant variance and the $lag - h$ covariances depend only on the time interval $h$ i.e. the process $\{X_i\}$ is covariance stationary when:

1. $E[X_i] = \mu_x$,
2. $\sigma^2[X_i] = E[X_i^2] - \mu_x^2$,
3. $Cov\{X_i, X_{i+h}\} = E[X_i X_{i+h}] - \mu_x^2 = R(h)$.

## 4.2 Initialization Detection

If one simulates a stochastic process, which involves an input of random numbers, it is well known that such process is in the beginning in its transient phase and its stochastic parameters vary in time. It is important to use a transient detection method to delete observations from such stage and use only those observations $x_n$ that have been collected after the process passed this stage and is, therefore, stable. It can be said that the main reason one would be using initial transient detection method is to find a truncation point, after which observations can be considered as representing the steady-state of a stochastic process. It was shown that including a initial transient detection method with fixed sample size approach can improve the quality of and estimator, meaning its bias will be lowered [20]. In [17] McNickle mentiones that with sequential approach this is not the issue, as one could run the simulation long enough to reduce the influence of initial observations. However, in [17] McNickle has also found that it is recommended to use a method of transient detection, to prevent simulation stopping prematurely.

### 4.2.1 Schruben's Test

First method that has been used in the experiment is based on Schruben's test of stationarity, as implemented by Pawlikowski in [20], and its main idea is to test, sequentially, if sufficient number of initial observations has been deleted and the process can be assumed to be in its steady-state, hence observations $x_n$ used for the final estimation and analysis represent the steady-state parameters. The heuristic rule that "the initial transient period is over after $n_0$ observations if the time series $x_1, x_2, ..., x_{n_0}$ crosses the mean $\bar{X}_{n_0}$ $k$ times "[20] is applied to get the first approximation of a truncation point $n_0^*$. $\bar{X}_{n_0}$ is the mean estimate $\hat{\theta}(n)$, and $k$ are selected to be equal to 25. Using this rule we get a sequence $n_t$ of observations that can be tested for stationarity. As mentioned in [20] a problem arises that it is necessary to know the degrees of freedom for the steady-state estimator $\sigma^2(\hat{\theta}(n))$ earlier than we know that the process is in its stationary phase. The paper mentions that one should only do the estimation over observations from subsequence $n_v$ of currently testing sequence $n_t$. Taking $n_t \geq \gamma_v n_v$, where $\gamma_v \geq 2$. Schruben has selected the $n_t$ to be at least $n_t = 200$, meaning 200 observations would be stored before stationarity testing. However, the initial period can be longer than $n = 200$, therefore Pawlikowski recommended in [20] that:

$$n_t = max(\gamma_v n_v, \gamma n_o^*), \tag{4.2}$$

"where $\gamma n_0^*$ is the smallest length of one step in sequential testing for stationarity for a given $n_0^*, \gamma > 0$ [20]" . The method would discard $n_0^*$ observations and after next $n_t$ observations are collected it would calculate the $\sigma^2(\hat{\theta}(n))$ using one of the methods mentioned in Chapter 3. When the $\sigma^2(\hat{\theta}(n))$ is estimated the method can start testing the $n_t$ sequence for stationarity. If the truncation point was assumed correctly, the process can be assumed to be in its steady state. If not more $\gamma n_0^*$ observations would be collected and whole process would be repeated until the length of initial period has been found, or limit for maximum number of observations used for the detection is reached. If the maximum number is reached, process can be regarded as not stable. However, it was shown in [20] that Schruben's test is rarely initiated after the mean $\bar{X}_{n_0}$ is crossed 25 times and this point is declared by the end of initial transient period by the first Schruben's test. In our research we refer to this methods as "25 crossings".

For more information please refer to [20].

### 4.2.2 Method Of Cumulative Means

Second method used within the experiment is a transient detection method based on cumulative means and developed by Freeth in [6]. Cumulative mean is constructed from sequence of all observations $x_n$ and produces a new sequence of such means [6]:

$$C_t = \frac{1}{t+1} \sum_{i=0}^{t} x_n, \tag{4.3}$$

$C_t$ is then a running mean of $t+1$ observations at time $t$. Where the observations are expected to be realization of a system in its steady-state. However, as was shown before, if one runs the simulation for a long period of time, the effect of initial observations becomes negligible. That said it can be seen that $C_t$ would eventually converge to the steady-state mean $\mu_x$ as $t$ increases, from a graphical point of view $C_t$, the graph would become more and more flat for increasing $t$ [6]. Hence, it can be seen that cumulative means can be used as a method for detecting the truncation point in sequential steady-state simulations. To allow automatic detection of the truncation point a forecasting method has to be used to decide if the plot of $C_t$ has become sufficiently flat and horizontal.

Freeth has found that to detect the flatness of the plot of $C_t$ a single exponential smoothing is to be used instead of double exponential smoothing. That has been found on a basis of preliminary testing, as it is not necessary to use forecasting using slopes as double exponential smoothing uses. It is also due to that if $C_t$ converges to flatness single smoothing would be more precise and the bad accuracy while the system is not in its steady-state will be amplified, hence allowing for easier determination of the truncation point [6]. The smoothed time is represented by [6]:

$$s_t = \alpha C_t + (1-\alpha)s_{t-1}, t \geq 1, \tag{4.4}$$

where $s_0 = C_0$ and $\alpha$ is the smoothing factor such that $0 < \alpha < 1$, as Freeth has found lower the $\alpha$ is the bigger the smoothing is. To forecast subsequent $C_{t+1}$ the value of $s_t$ is used. Then one step ahead error $e_t$ forecasting is used to detect the truncation point. $e_t$ is given by:

$$e_t = s_{t-1} - C_t, \tag{4.5}$$

so then $e_t$ would converge to zero as $s_t$ converges to $C_t$. However, due to the randomness of the process, $s_t$ and $C_t$ could cross even within initial phase. A method for correctly detecting $e_t$ has to be used.

Using absolute error of $e_t$ is not sufficient enough as it could give underestimated value of truncation point. To overcome this, Freeth proposed a detection condition based on the sample standard deviation of the forecasting errors $S_e$:

$$E_t \leq \gamma N S_e, \tag{4.6}$$

The stopping condition compares $E_t$ with the variation in the observed data, $\gamma$, $\gamma > 0$, is a constant, all the observations within this window can be assumed to be characterizing the steady-state if such condition is met and the truncation point is then a point at the beginning of this sequence $l = i - N + 1$ [6]. It has been experimentally found that $E_t$ should be calculated from sum of squared forecasting errors to reduce the influence of $S_e$ being taken from non-stationary phase, which would artificially widen the value of $E_t$. $E_t$ is then:

$$E_t = \sum_{i=0}^{N-1} e_{t-1}^2, t \geq N - 1, \tag{4.7}$$

The values of $\alpha$ and $\gamma$ have been found experimentally to perform the best at 0.01 and 0.1 respectively.

Since we are dealing with steady-state simulations in this thesis we propose that Cumulative Means should be used as an initial transient detection method, as it produces much longer length of initial transient and is, therefore, safer than “25 crossings” method [6]. So that one can be safer in assuming that observations $x_n$ used for the estimation of the steady-state mean are actually observations characterizing the steady-state behaviour of a stochastic process being simulated. On the other hand, as it is shown in [17], the selection of method used for the initial transient detection does not have a huge impact on the final steady-state estimates as the simulation runs long enough.

# Analysis

# Chapter 5

# Coverage Analysis

As mentioned above, to be confident that simulation is producing the correct results, one needs to be sure that the model used for simulation is valid and representing correctly the real world system under study. Secondly, a correct pseudo random generator has to be used, PRNGs were be introduced in Section 2.6. Lastly, use a statistically correct method of output analysis. This thesis deals with the last problem i.e. how good is an estimator in terms of its coverage by CIs. We have been focusing on steady-state mean analysis of stochastic processes under sequential approach to simulation.

There are three common measurements used to assess the quality of an estimator [20]:

1. Bias of an estimator, is a measure of systematic error of measured parameter, steady-state mean in our case. Or difference between the estimated value to the true mean.

$$Bias[\hat{\theta}(n)] = E[\hat{\theta}(n) - \mu_x]. \tag{5.1}$$

2. Variance of an estimator, is a squared deviation of the estimator from its mean value.

$$\sigma^2[\hat{\theta}(n)] = E[\{\hat{\theta}(n) - E[\hat{\theta}(n)]\}^2]. \tag{5.2}$$

3. Mean squared error, is the difference between the estimator and what is estimated. MSE corresponds to the expected value of the squared error. The difference occurs due to the randomness or because the estimator is not correct in its assumptions, in our case does not address the autocorrelation of data correctly.

$$MSE[\hat{\theta}(n)] = Bias[\hat{\theta}(n)]^2 + \sigma^2[\hat{\theta}(n)]. \tag{5.3}$$

The main problem with these measures is that if the estimator does not address the autocorrelation correctly (every estimator contains different assumptions), the MSE value will be affected depending on the quality of the method. Also, as it can be seen from the Equation 5.3 if variance grows and bias decreases, MSE remains low, but the final CIs produced might be bigger and unstable.

Therefore a different approach to the assessment of quality of an estimator has to be developed. Pawlikowski et al. in [22] proposed an experimental approach to the assessment of the quality of an estimator. The coverage of confidence intervals, as mentioned above, is defined as a relative frequency with which the final confidence interval contains the true value $\mu_x$. If one sets up the theoretical confidence level to be $1-\alpha$ it is also expected that if simulation is run 100 times, the CIs would cover the true mean $1-\alpha$ times. However, this is not always true in practice. The coverage can be determined together with its CI [22]:

$$\left(c - z_{1-\frac{\alpha}{2}}\sqrt{\frac{c(1-c)}{n_c}}, c + z_{1-\frac{\alpha}{2}}\sqrt{\frac{c(1-c)}{n_c}}\right), \tag{5.4}$$

where $c$ is the estimated coverage of confidence intervals and $n_c$ is the number of replicated coverage experiments. One would expect $c$ to be very close to the theoretical level $1-\alpha$. Pawlikowski in the article argues that, if sequential approach was used to produce one coverage experiment, the coverage analysis has to be done in sequential way as well. Three rules have been set up in [22]:

1. Coverage should be analysed sequentially, i.e. analysis of coverage should be stopped when relative precision (the relative half-width of confidence interval) of the estimated coverage satisfies a specified level.
2. An estimate of coverage has to be calculated from a representative sample of data, so the coverage analysis can start only after minimum number of "bad" confidence intervals have been recorded.
3. Results from simulation runs that are clearly too short should not be taken into account.

We have considered methods of MSE-DPBM [28] and SA/HW [16] for the coverage analysis, as SA/HW has been already shown to perform better in terms of coverage of confidence intervals to sequential method of NBM [20] in [22]. Method of MSE-DPBM has been selected on basis that it is believed to be superior to other batch means methods and is the latest one in evolution of estimator based on BM.

Using experimental analysis of coverage of confidence intervals limits us to use of analytically tractable models only, because a true mean $\mu_x$ has to be known in order to assess if CI covers the $\mu_x$ or not. Following section will introduce the stochastic processes that have been used for the coverage analysis. We have focused solely on mean waiting time, $\theta_w$, analysis of the queueing processes and mean value of autoregressive process. These processes have been selected on a basis of their autocorrelation functions, where queueing processes have non-fluctuating shape and autoregressive processes have fluctuating autocorrelation function and are used widely as reference models.

## 5.1 M/G/1

A M/G/1 process is described here only for description purposes of queueing processes. M/G/1 are queuing processes where the arrival times are Poisson distributed, service times are of general distribution and one server is present. Traffic intensity $\rho$ is given by $\rho = \frac{\lambda}{\mu}$, where $\lambda$ is arrival rate and $\mu$ is the service rate. As this experiment is dealing only with processes that are stable, condition $\rho < 1$ has to be satisfied. The service time has been set to $\mu = 10$ as most of the processes are already implemented in Akaroa2, specifically M/M/1, M/D/1 and M/H$_2$/1. The processes were initialized idle.

## 5.2 M/M/1

A M/M/1 process is a M/G/1 process with exponentially distributed service rates. The steady-state mean for mean waiting time, the time customer spends in the queue is given by [12]:

$$\theta_w = \frac{\rho}{\lambda(1-\rho}. \tag{5.5}$$

## 5.3 M/D/1

A M/D/1 process is a M/G/1 process with deterministic service rates D($mu = 1/D$). The steady-state mean for mean waiting time [12]:

$$\theta_w = \frac{1}{2\mu}\frac{\rho}{1-\rho}. \tag{5.6}$$

## 5.4 M/H$_2$/1

A M/H$_2$/1 processes have Erlang distributed, with shape parameter k=2, service rates. The steady-state mean for mean waiting time [12]:

$$\theta_w = \rho - \frac{\rho^2}{8(1-\rho)}. \tag{5.7}$$

## 5.5 Autoregressive Process

An autoregressive process of order one, AR(1), have output generated by a formula [7]:

$$X_t = c + \phi X_{t-1} + \epsilon_t, \tag{5.8}$$

for a constant $c$ and $\phi$, which is the parameter of the autocorrelation model. The white noise $\epsilon_t$ is a Gaussian distributed with zero mean $E[\epsilon_t] = 0$ and constant variance $\sigma^2[\epsilon_t] = \sigma_\epsilon^2$. The mean is then calculated as following:

$$\mu = \frac{c}{1-\phi}. \tag{5.9}$$

AR(1) was implemented for this thesis as following, $c = 1$, and $\epsilon_t$ was taken as $N(0,1)$ random number. Parameter $c$ was selected to be equal to one, because of the relative error calculation in Equation 2.6, to avoid division by zero.

## 5.6 Open Queueing Network

The open queuing network was implemented according to the figure 5.1.

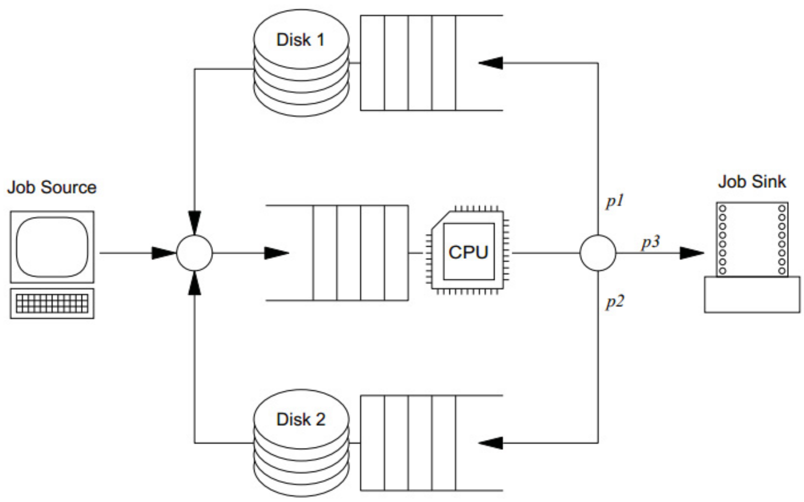


Figure 5.1: Open Queueing network

After processing a random fraction of jobs $p_1$, $p_2$ return to Disk 1 or Disk 2. A fraction $p_3$ leaves the system. The mean CPU service time is 6, the mean service time for each disk is $p_1 = p_2 = 0.4$, all distributions are negative exponential, and the source rate is set to give loads at the CPU ranging from 0.5 to 0.95 [17]. This model is referred to as QNet later in this thesis.

## 5.7 Implementation of Coverage Analysis

. Coverage analysis has been introduced in the previous section of this chapter. We use the rules as specified in [22] to evaluate the correctness of simulation output analysis methods (SOAM), such as SA/HW, DPBM, MSE-DPBM and Modified MSE-DPBM, in simulations of analytically tractable models. This allows us to compare the experimental, estimated, value to the true performance parameter of a model, in our case we are only considering steady-state mean $\mu_x$. Coverage is the experimental confidence level of the final CIs produced by a given SOAM. Meaning the proportion of final experimental CIs that cover the true mean of a model, see 5.4. We are using a sequential approach to the coverage analysis, and such analysis is stopped when absolute error $\Delta_{z-\frac{1}{2}}$ is lower than 1 %. Since we expect the values to be very close to one, using theoretical confidence level of 95%, we can use the absolute error instead of relative error as in Equation 2.6 and be safe that the CIs are narrow and stable.

It is often not true that in practice that a method would cover the true mean $\mu_x$ at the expected confidence level $1 - \alpha = 95\%$, therefore we accept the SOAM as being correct if the experimental coverage is sufficiently close to the theoretical value. If the coverage is higher than expected we assume that the SOAM might be stopping the simulation too soon. If the coverage is lower the SOAM is most likely stopping the simulation too early.

For the coverage analysis we use the models described above. We use wide range of traffic intensities, $\rho$, for the queueing models $(0.5, 0.6, ..., 0.95)$ and the same range for the AR(1)'s $\phi$ parameter. All the queueing models have been initialized empty with service rate $\mu = 10$. We have selected the memory size parameter of DPBM $k, k = 15$ and 50 for MSE-DPBM $k = 15$ and for Modified MSE-DPBM $k = 15$.

Implementation of the rules as specified in [22]:

- Record 200 "bad" CIs, the CIs that do not cover the true mean of a simulated model.
- Compute the average length from all simulations $\bar{L}$ and standard de-

viation $\sigma(\bar{L})$.

- Reject all simulation runs that are shorter than one standard deviation below the mean $L_{min} = \bar{L} - \sigma(\bar{L})$.
- Compare estimated $\hat{\theta}$ to the true mean $\mu_x$. Continue until stopping condition has been reached $\Delta_{z-\frac{1}{2}} < 1$, while still rejecting the too short simulations.

As the study is very intensive on time and computational resources a MySQL database is set up, where we keep all the results of each coverage experiment (single simulation experiment) separate. Block diagram of the coverage analysis can be see in Figure 5.2. Coding is done in Perl and can be seen in Appendix D.

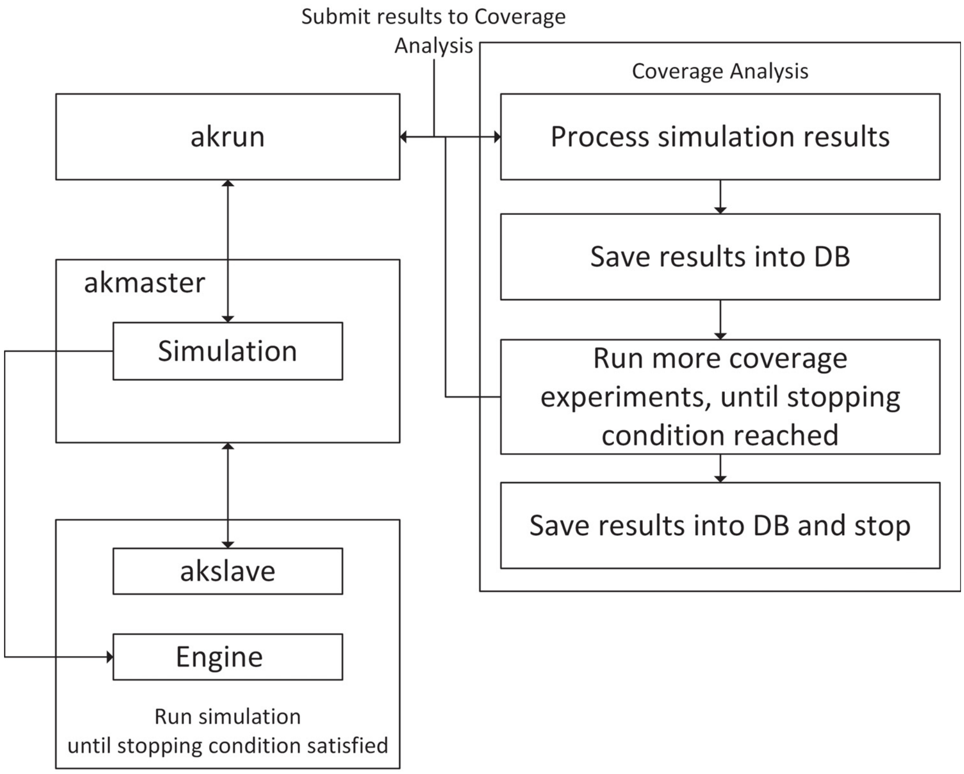


Figure 5.2: Coverage Analysis

Coverage has been run for all the combinations as can be seen in the Table 5.1.

We have used 64bit Linux distribution with CPUs based on Intel architecture for the experiment. We use a single processor scenario of MRIP to assess the quality of methods.

| Model | Analysis Method | Initial Transient | Load/$\phi$ | Confidence level |
|---|---|---|---|---|
| M/M/1 | DPBM | Schruben | 0.5, 0.6 , ...,0.95 | 0.95 |
| M/M/1 | DPBM | CumulativeMeans | 0.5, 0.6 ...,0.95 | 0.95 |
| M/M/1 | SA/HW | Schruben | 0.5, 0.6 , ...,0.95 | 0.95 |
| M/M/1 | SA/HW | CumulativeMeans | 0.5, 0.6 ...,0.95 | 0.95 |
| M/$H_2$/1 | DPBM | Schruben | 0.5, 0.6 , ...,0.95 | 0.95 |
| M/$H_2$/1 | DPBM | CumulativeMeans | 0.5, 0.6 ...,0.95 | 0.95 |
| M/$H_2$/1 | SA/HW | Schruben | 0.5, 0.6 , ...,0.95 | 0.95 |
| M/$H_2$/1 | SA/HW | CumulativeMeans | 0.5, 0.6 ...,0.95 | 0.95 |
| QNet | DPBM | Schruben | 0.5, 0.6 , ...,0.95 | 0.95 |
| QNet | DPBM | CumulativeMeans | 0.5, 0.6 ...,0.95 | 0.95 |
| QNet | SA/HW | Schruben | 0.5, 0.6 , ...,0.95 | 0.95 |
| QNet | SA/HW | CumulativeMeans | 0.5, 0.6 ...,0.95 | 0.95 |
| M/D/1 | DPBM | Schruben | 0.5, 0.6 , ...,0.95 | 0.95 |
| M/D/1 | DPBM | CumulativeMeans | 0.5, 0.6 ...,0.95 | 0.95 |
| M/D/1 | SA/HW | Schruben | 0.5, 0.6 , ...,0.95 | 0.95 |
| M/D/1 | SA/HW | CumulativeMeans | 0.5, 0.6 ...,0.95 | 0.95 |
| AR(1) | DPBM | Schruben | 0.5, 0.6 , ...,0.95 | 0.95 |
| AR(1) | DPBM | CumulativeMeans | 0.5, 0.6 ...,0.95 | 0.95 |
| AR(1) | SA/HW | Schruben | 0.5, 0.6 , ...,0.95 | 0.95 |
| AR(1) | SA/HW | CumulativeMeans | 0.5, 0.6 ...,0.95 | 0.95 |

Table 5.1: Coverage Experiments

# Experiment Design

# Chapter 6

# Experiment Design

In this chapter we introduce the experiment design and settings. We present the modifications of DPBM methods, this are done in order to be able to implement such methods as a component of Akaroa2.

## 6.1 Akaroa2 Simulation Controller

The methods of DPBM have to be modified for their inclusion as a component of Akaroa2. This functions are receive an observation, check if checkpoint was reached and send the estimated values $\hat{\theta}(n)$ and $\sigma^2(\hat{\theta})$. The method then runs on each of *akslave*'s simulation engines and performs the local estimation of parameters, these estimations are in our case: mean $\hat{\theta}$ and variance of the mean $\sigma^2(\hat{\theta})$. Methods have to register themselves as a available method to Akaroa2, see [4] or Appendix F. It is also necessary to specify checkpoint spacing, a point in time when estimates are sent to *akmaster* for analysis. Checkpoint spacing will be introduced in Section 6.2.1.

## 6.2 Output Analysis Methods

Four output analysis methods are selected for the analysis of their coverage as introduced in Chapter 5. Namely SA/HW as implemented in [16] by Pawlikowski et al., DPBM [27], MSE-DPBM [28], as implemented by Song, and a modified version of MSE-DPBM. All of the methods, except SA/HW which is already present in Akaroa2, are presented here in their modified form. The modifications include functions that are necessary in order for the methods to work as a component of Akaroa2.

### 6.2.1 Dynamic Partially-Overlapping Batch Means

Dynamic partially-overlapping batch means as implemented in [27] by Song, is selected for the experiment only as a reference method, and is not suitable for automated analysis in Akaroa2. Every method, to be considered automated, can not have batch size $m$ or number of batches $k$ fixed before the simulation run. Both need to be dynamically selected during the run-time of the simulation experiment. And as it flows from DPBM [27], the memory parameter $k$ (number of batches) is fixed. The methods from Akaroa2, that are included are:

- CheckpointReached(), function to decide if checkpoint was reached.
- GetCheckpoint(Checkpoint &cp), if checkpoint is reached, estimates are send to Akaroa2's *akmaster*, to check if stopping condition was reached.
- ProcessObservation(real value), starts to process observation, if any, that comes from simulation engine. Value is the observation $x_n$.

We make changes to the algorithm that is implemented in [27], as it is inconsistent with implementation of MSE-DPBM [28]. The vector $L(i) = 1, 2, ..., 8k$ consists of four sub-vectors $A, B, C, D$. In implementation [27] the vectors are updated with new observation $x_n$ every time a observations comes, on the other hand in [28] the new observation is saved into the vector $B$ only after one, $h > 0$, collapsing has occurred, respectively to $B, C$ and $D$ after at least two collapses, $h > 1$, have occurred. As we have found experimentally this updating does not make any difference to the estimates $\hat{\theta}(n)$ and $\sigma^2\theta$. Therefore, as a baseline the updating as in [28] is used for the implementation here. As we have coded the DPBM method first and implemented estimation of $\hat{m}^*$ on top of that code implementation is as following for all the methods (DPBM, MSE-DPBM and modified MSE-DPBM):

- Memory parameter $g$ is named $k$.
- Changed naming for $r_A = r_1$, $r_B = r_2$, $r_C = r_3$, $r_D = r_4$ and similarly for $m_A...m_D = m_1, ..., m_4$ to keep the naming conventions same for all the methods.
- The updating of B, C and D is done as explained above.
- The checkpoint spacing has been selected as follows: CheckpointReached() can return true iff collapsing has occurred $h > 1$ and new batch has been collected $m_1 = m$.
- Code implementation:

- $k$ = memorySize.
- $h$ = numberOfCollapsingOccured.
- $n$ = currentSampleSizeN.
- $\hat{V}_{DPBM}(m)$ = estimatorDPBM.
- $m$ = batchSize.

The algorithm as can be seen in Figure A.1:

1. Start and initialize variables, $n = 1,\ h = 0,\ m = 1,\ L(i) = 0,\ i = (1,\ 2,\ ...,\ 8k),\ m_j = 0,\ r_j = 0,\ j = 1,\ 2,\ 3,\ 4$.

2. Wait for observation $x_n$ from Akaroa2 simulation engine and start processing it.

3. If current cell in vector $A$ has room increase batch size $m$ += 1 and go to step 7, else go to step 4.

4. Does vector $A$ has room i.e. $r_1 < 2k$? If yes go to step 4.1, or go to step 4.2.

   4.1. Initialize $m_1 = 1$ and set $r_1+ = 1$ to point to the next cell. Go to step 6.

   4.2. If $h = 0$ go to 4.2.1, else go to 4.2.2.

   4.2.1. Collapse the vectors $A$ and $B$ and initialize values of $m_1,\ m_2,\ r_1,\ r_2$
   - $B(i) = A(2i) + A(2i+1), i = 1,\ ...,\ g-1;\ B(k) = A(2k)$.
   - $A(i) = A(2i-1) + A(2i), i = 1,\ ...,\ g$.
   - $m_1 = 1,\ m_2 = 2^h,\ r_1 = k+1,\ r_2 = g$.
   - Go to step 5.

   4.2.2. Collapse the vectors to $D,\ C,\ B,\ A$ and initialize values of $m_1,\ m_2,\ m_3,\ m_4,\ r_1,\ r_2,\ r_3,\ r_4$.
   - $D(i) = B(2i) + B(2i+1), i = 1,\ ...,\ k-1;\ D(k) = B(2k)$.
   - $C(i) = B(2i-1) + B(2i), i = 1,\ ...,\ k$.
   - $B(i) = A(2i) + A(2i+1), i = 1,\ ...,\ k-1;\ B(k) = A(2k)$.
   - $A(i) = A(2i-1) + A(2i), i = 1,\ ...,\ k$.
   - $m_1 = 1,\ m_2 = 2^h,\ m_3 = 2^k + 2^{k-1},\ m_4 = 2^{k-1}$.
   - $r_1 = k+1,\ r_2 = k,\ r_3 = k,\ r_4 = k$.
   - Go to step 5.

5. Update $h$ += 1 and $m = 2^h$.

6. Initialize the sum stored in vector $A_1 = 0$.

7. Add the new observations $x_n$ in the current cell in $A$, $A(r_1)$ += $x_n$.

8. If number of collapsing $h > 0$ go to step 8.1, else go to step 9.
   - 8.1. If $h = 1$ go to step 8.2, else go to step 8.3.
   - 8.2. If cell $B(r_2)$ has room $(m_2 < m)$, set $m_2 \mathrel{+}= 1$, else set $m_2 = 1, r_2 \mathrel{+}= 1, B(r_2) = 0$. Update $B(r_2) \mathrel{+}= x_n$. Go to step 9.
   - 8.3. If cell $B(r_2)$ has room $(m_2 < m)$, set $m_2 \mathrel{+}= 1$, else set $m_2 = 1, r_2 \mathrel{+}= 1, B(r_2) = 0$. Update $B(r_2) \mathrel{+}= x_n$.
   - 8.4. If cell $C(r_3)$ has room $(m_3 < m)$, set $m_3 \mathrel{+}= 1$, else set $m_3 = 1, r_3 \mathrel{+}= 1, C(r_3) = 0$. Update $C(r_3) \mathrel{+}= x_n$.
   - 8.5. If cell $D(r_4)$ has room $(m_4 < m)$, set $m_4 \mathrel{+}= 1$, else set $m_4 = 1, r_4 \mathrel{+}= 1, D(r_4) = 0$. Update $D(r_4) \mathrel{+}= x_n$. Go to step 9.
9. Increase the sample size $n \mathrel{+}= 1$.
10. Call checkpointReached(), if checkpoint was reached go to step 11, else go to step 2.
11. Calculate $\hat{\theta}(n)$, $\sigma^2(\hat{\theta})$ and send these values to *akmaster*.
12. If stopping condition satisfied, stop simulation and present estimated results, else go to step 2.

Flowchart A.1 and C++ code of the method can be found in Appedix A.

### 6.2.2 MSE-DPBM

Optimal mean squared error DPBM provides the facility to dynamically change number of batches $k$ and batch size $m$ during the run-time of a simulation, as shown in Section 3.2.5. MSE-DPBM is implemented accordingly to [28] with changes as introduced in Section 6.2.1. The estimators for $\hat{m}^*$ 3.9, sum of all correlations 3.10 and sum of all weighted correlations 3.11 assume that the sample size $n$ and batch size $m$ are large enough, see Section 3.1 in [28]. When the simulation starts the sample size and batch size are equal to $n = 1$ and $m = 2^h = 1$ respectively, where $h$ is number of collapses. It can be seen that with first collapsing and calculation of $\hat{m}^*$ the batch size and sample size are $m = 2$ and $n = m \times 2k$ respectively. Therefore if user selects the memory size to be 1 ($k = 1$), $m$ and $n$ are not "large enough" as $m = 2$ and $n = 2$. The estimation of $\hat{V}_{DPBM}(m)$ occurs every time the method for estimation $\hat{m}^*$ is called. For implementation of this method we use the same methods from Akaroa2 as in DPBM section imp:dpbm. Additional variables that are included in MSE-DPBM:

- $\hat{V}_B(m/2)$ = previousDPBM.
- $\hat{m}^*$ = optimalBatchSize.

- Checkpoint spacing is implemented in the same way as in DPBM.

Algorithm for MSE-DPBM, as implemented in Akaroa2, follows:

1. Start and initialize variables, $n = 1,\ h = 0,\ m = 1,\ L(i) = 0,\ i = (1,\ 2,\ ...,\ 8k),\ m_j = 0,\ r_j = 0,\ j = 1,\ 2,\ 3,\ 4.$

2. Wait for observation $x_n$ from Akaroa2 simulation engine and start processing it.

3. If current cell in vector $A$ has room increase batch size $m$ += 1 and go to step 7, else go to step 4.

4. Does vector $A$ has room i.e. $r_1 < 2k$? If yes go to step 4.1, or go to step 4.2.

   4.1. Initialize $m_1 = 1$ and set $r_1 + = 1$ to point to the next cell. Go to step 6.

   4.2. If $h = 0$ go to 4.2.3, else go to 4.2.1.

   4.2.1. Compute $\hat{m}^*$, i.e. compute $\hat{V}_B(m/2)$ [3.13], $\hat{V}_{DPBM}(m)$ [3.7] and estimate $\hat{m}* = (1.03n(\frac{\hat{\gamma}_1}{\hat{\gamma}_2})^2)^{\frac{1}{3}} + 1$ [3.9].

   4.2.2. If $m < \hat{m*}$ go to step 4.2.4, else increase memory $k$ += 1 and go to step 4.

   4.2.3. Collapse the vectors $A$ and $B$ and initialize values of $m_1,\ m_2,\ r_1,\ r_2$
   - $B(i) = A(2i) + A(2i+1), i = 1,\ ...,\ g-1;\ B(k) = A(2k).$
   - $A(i) = A(2i-1) + A(2i), i = 1,\ ...,\ g.$
   - $m_1 = 1,\ m_2 = 2^h,\ r_1 = k+1,\ r_2 = g.$
   - Go to step 5.

   4.2.4. Collapse the vectors to $D,\ C,\ B,\ A$ and initialize values of $m_1,\ m_2,\ m_3,\ m_4,\ r_1,\ r_2,\ r_3,\ r_4.$
   - $D(i) = B(2i) + B(2i+1), i = 1,\ ...,\ k-1;\ D(k) = B(2k).$
   - $C(i) = B(2i-1) + B(2i), i = 1,\ ...,\ k.$
   - $B(i) = A(2i) + A(2i+1), i = 1,\ ...,\ k-1;\ B(k) = A(2k).$
   - $A(i) = A(2i-1) + A(2i), i = 1,\ ...,\ k.$
   - $m_1 = 1,\ m_2 = 2^h,\ m_3 = 2^k + 2^{k-1},\ m_4 = 2^{k-1}.$
   - $r_1 = k+1,\ r_2 = k,\ r_3 = k,\ r_4 = k.$
   - Go to step 5.

5. Update $h$ += 1 and $m = 2^h$.

6. Initialize the sum stored in vector $A_1 = 0$.

7. Add the new observations $x_n$ in the current cell in $A$, $A(r_1)$ += $x_n$.

8. If number of collapsing $h > 0$ go to step 8.1, else go to step 9.

8.1. If $h = 1$ go to step 8.2 else go to step 8.3.

8.2. If cell $B(r_2)$ has room ($m_2 < m$), set $m_2$ += 1, else set $m_2 = 1, r_2$ += $1, B(r_2) = 0$. Update $B(r_2)$ += $x_n$. Go to step 9.

8.3. If cell $B(r_2)$ has room ($m_2 < m$), set $m_2$ += 1, else set $m_2 = 1, r_2$ += $1, B(r_2) = 0$. Update $B(r_2)$ += $x_n$.

8.4. If cell $C(r_3)$ has room ($m_3 < m$), set $m_3$ += 1, else set $m_3 = 1, r_3$ += $1, C(r_3) = 0$. Update $C(r_3)$ += $x_n$.

8.5. If cell $D(r_4)$ has room ($m_4 < m$), set $m_4$ += 1, else set $m_4 = 1, r_4$ += $1, D(r_4) = 0$. Update $D(r_4)$ += $x_n$. Go to step 9.

9. Increase the sample size $n$ += 1.

10. Call checkpointReached(), if checkpoint was reached go to step 11, else go to step 2.

11. Calculate $\hat{\theta}(n)$, $\sigma^2(\hat{\theta}(n))$ and send these values to akmaster.

12. If stopping condition satisfied, stop simulation and present estimated results, else go to step 2.

Flowchart B.1 and C++ code of the method can be found in Appedix B.

### 6.2.3 Modified MSE-DPBM

We have discussed the "large enough" problem in the Section 6.2.2. In this section we introduce the changes to the MSE-DPBM algorithm.

It seems unnecessary to calculate $\hat{V}_B(m/2)$ every time that vector $A$ has become full to estimate the $\hat{m}^*$, especially at the beginning of the simulation when $n$ and $m$ are low. Our implementation does not use the estimation of $\hat{V}_B(m/2)$ to estimate the optimal batch size, rather we save the value of current $V_{DPBM}(m)$ every time that estimation of $\hat{m}^*$ occurs, when we come to the next estimation the value of $V_{DPBM}(m)$ is available and can be used as $V_B(m/2)$, we call this as $V_{DPBM}(m/2)$. In addition to that we introduce a waiting period, a period when no estimation of $\hat{m}^*$ occurs. This period is present until the batch size is greater than 64, $m > 64$. In this period we just collapse the vectors accordingly and compute $V_{DPBM}(m)$ to be used in the first estimation of $\hat{m}^*$ as $V_B(m/2)$. The value of $m > 64$ was selected on the basis that if memory size $k = 1$ is selected, we would have at least 128 observations available for the calculation of CI, see Section 2.3 for the discussion of normal distribution. It has been shown in [28] that $V_{DPBM}(m/2)$ and $V_B(m/2)$ are assymptotically equal even though $V_B(m/2)$ is only 50 % OBM compared to 75 % OBM of $V_{DPBM}(m/2)$. The algorithm is presented next:
Algorithm for modified version of MSE-DPBM:

1. Start and initialize variables, $n = 1,\ h = 0,\ m = 1,\ L(i) = 0,\ i = (1,\ 2,\ ...,\ 8k),\ m_j = 0,\ r_j = 0,\ j = 1,\ 2,\ 3,\ 4.$

2. Wait for observation $x_n$ from Akaroa2 simulation engine and start processing it.

3. If current cell in vector $A$ has room increase batch size $m$ += 1 and go to step 7, else go to next step.

4. Does vector $A$ has room i.e. $r_1 < 2k$? If yes go to step 4.1, or go to step 4.2.

   4.1. Initialize $m_1 = 1$ and set $r_1+ = 1$ to point to the next cell. Go to step 6.

   4.2. If $h = 0$ go to 4.2.4, else go to 4.2.1.

   4.2.1. If $m > 64$ go to next step, else go to 4.2.5.

   4.2.2. Compute $\hat{m}^*$, i.e. use previous $\hat{V}_{DPBM}(m/2)$ and $\hat{V}_{DPBM}(m)$ [3.7] and estimate $\hat{m}* = (1.03n(\frac{\hat{\gamma}_1}{\hat{\gamma}_2})^2)^{\frac{1}{3}} + 1$ [3.9].

   4.2.3. If $m < \hat{m}^*$ go to step 4.2.5, else increase memory $k$ += 1 and go to step 4.

   4.2.4. Collapse the vectors $A$ and $B$ and initialize values of $m_1,\ m_2,\ r_1,\ r_2$
   - $B(i) = A(2i) + A(2i+1), i = 1,\ ...,\ g-1;\ B(k) = A(2k).$
   - $A(i) = A(2i-1) + A(2i), i = 1,\ ...,\ g.$
   - $m_1 = 1,\ m_2 = 2^h,\ r_1 = k+1,\ r_2 = g.$
   - Go to step 5.

   4.2.5. Collapse the vectors to $D,\ C,\ B,\ A$ and initialize values of $m_1,\ m_2,\ m_3,\ m_4,\ r_1,\ r_2,\ r_3,\ r_4.$
   - $D(i) = B(2i) + B(2i+1), i = 1,\ ...,\ k-1;\ D(k) = B(2k).$
   - $C(i) = B(2i-1) + B(2i), i = 1,\ ...,\ k.$
   - $B(i) = A(2i) + A(2i+1), i = 1,\ ...,\ k-1;\ B(k) = A(2k).$
   - $A(i) = A(2i-1) + A(2i), i = 1,\ ...,\ k.$
   - $m_1 = 1,\ m_2 = 2^h,\ m_3 = 2^k + 2^{k-1},\ m_4 = 2^{k-1}.$
   - $r_1 = k+1,\ r_2 = k,\ r_3 = k,\ r_4 = k.$
   - Go to step 5.

5. Update $h$ += 1 and $m = 2^h$.

6. Initialize the sum stored in vector $A_1 = 0$.

7. Add the new observations $x_n$ in the current cell in $A$, $A(r_1)$ += $x_n$.

8. If number of collapsing $h > 0$ go to step 8.1, else go to step 9.

   8.1. If $h = 1$ go to step 8.2 else go to step 8.3.

8.2. If cell $B(r_2)$ has room $(m_2 < m)$, set $m_2$ += 1, else set $m_2 = 1, r_2$ += $1, B(r_2) = 0$. Update $B(r_2)$ += $x_n$. Go to step 9.

8.3. If cell $B(r_2)$ has room $(m_2 < m)$, set $m_2$ += 1, else set $m_2 = 1, r_2$ += $1, B(r_2) = 0$. Update $B(r_2)$ += $x_n$.

8.4. If cell $C(r_3)$ has room $(m_3 < m)$, set $m_3$ += 1, else set $m_3 = 1, r_3$ += $1, C(r_3) = 0$. Update $C(r_3)$ += $x_n$.

8.5. If cell $D(r_4)$ has room $(m_4 < m)$, set $m_4$ += 1, else set $m_4 = 1, r_4$ += $1, D(r_4) = 0$. Update $D(r_4)$ += $x_n$.

8.6. If $m \leq 64$ save $\hat{V}_{DPBM}(m/2) = V_{DPBM}(m)$, and compute new $\hat{V}_{DPBM}(m)$, else go to step 9.

9. Increase the sample size $n$ += 1.

10. Call checkpointReached(), if checkpoint was reached go to step 11, else go to step 2.

11. Calculate $\hat{\theta}(n)$, $\sigma^2(\hat{\theta}(n))$ and send these values to akmaster.

12. If stopping condition satisfied, stop simulation and present estimated results, else go to step 2.

See Section MSE-DPBM in Appendix C for the C++ code and flowchart of this implementation.

# Results

# Chapter 7

# Results

In this chapter we present results of the conducted experiments. First, we introduce an experiment that tests the experimental average run length, number of observation per independent coverage simulation, compared to the theoretical average number of observations that are necessary to produce CIs with confidence level $1 - \alpha = 0.95$. Second experiment will introduce the coverage analysis of all the models, we first test the DPBM modification and select the one performing the best. Then we compare this method to SA/HW and decide if one performs better than the other. In the third experiment we compare the memory requirements of MSE-DPBM and Mod. MSE-DPBM and show the requirements for SA/HW as a reference, as well. Next section of Experiment 3 shows the average number of runs per simulation model and method of output analysis, the comparison is done to show the computational efficiency of such methods. In every experiment we compare the two initial transient detection methods, 25 crossings and Cumulative Means. This comparison is only done for a reference purposes, as it was said in Section 4.2.2 we recommend using Cumulative Means as a method of initial transient detection, as it detects longer transient periods and is, therefore, safer to use [6].

## 7.1 Experiment 1: Average Run Length

In this experiment we are assessing the average run length $\bar{L}$ of single simulation run, or number of observation, that has been collected during a coverage analysis experiment of selected models under various loads. We compare these experimental results to the theoretical results presented in Table 7 in [17]. McNickle has calculated the theoretical average number of observations, that is necessary to collect in order to obtain experimental confidence level of 0.90 and 0.95 respectively. We are doing the analysis for our implementation of DPBM, MSE-DPBM, Modified MSE-DPBM and SA/HW and decide if these methods are usable for coverage analysis under Akaroa2. The analysis is done for values of load $\rho, \rho = 0.5, 0.6, ...0.9$, as these are present for comparison in [17]. The selected models for the comparison are M/M/1, M/D/1 and M/$H_2$/1, models that have been theoretically evaluated by McNickle. Experiment results are going to tell us if used method is suitable for using for coverage analysis and for analysing the output of stochastic steady-state simulations. We have plotted the average number, average run length, of observations necessary to construct a CI of 0.95 confidence level on the x-axis, the y-axis is the number of observations, or length. As errors bars we have used the standard deviation of the length $\sigma(\bar{L})$. Refer to Tables E.2, E.3 and E.4 for exact results.

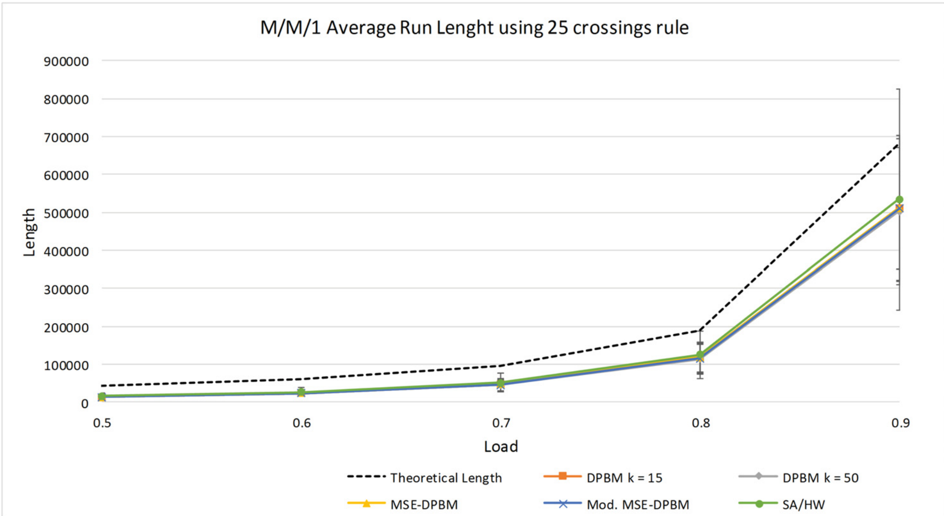


Figure 7.1: M/M/1's average run length per simulation using 25 crossings rule

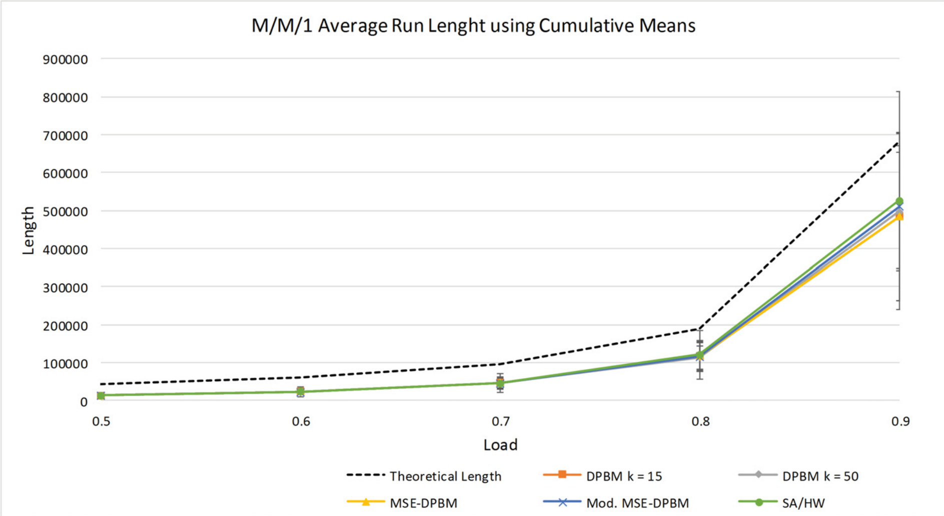


Figure 7.2: M/M/1's average run length per simulation using Cumulative Means

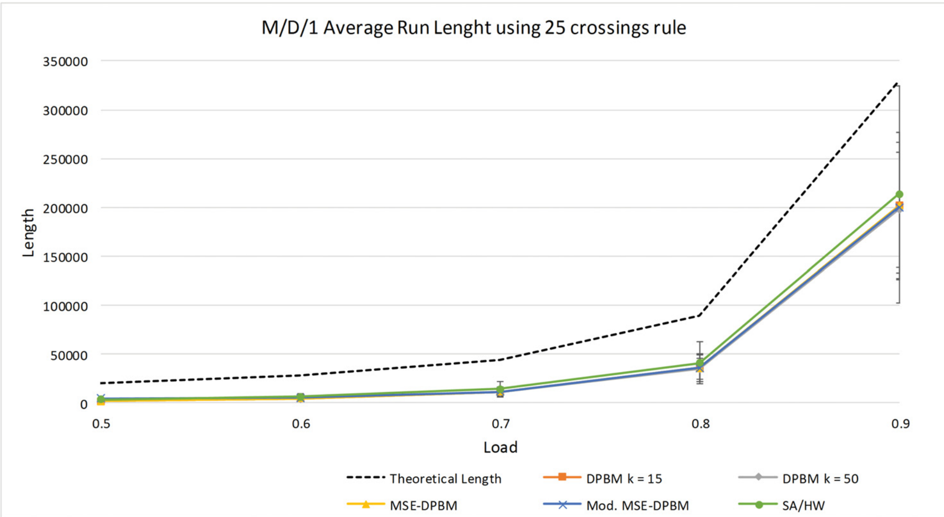


Figure 7.3: M/D/1's average run length per simulation using 25 crossings rule

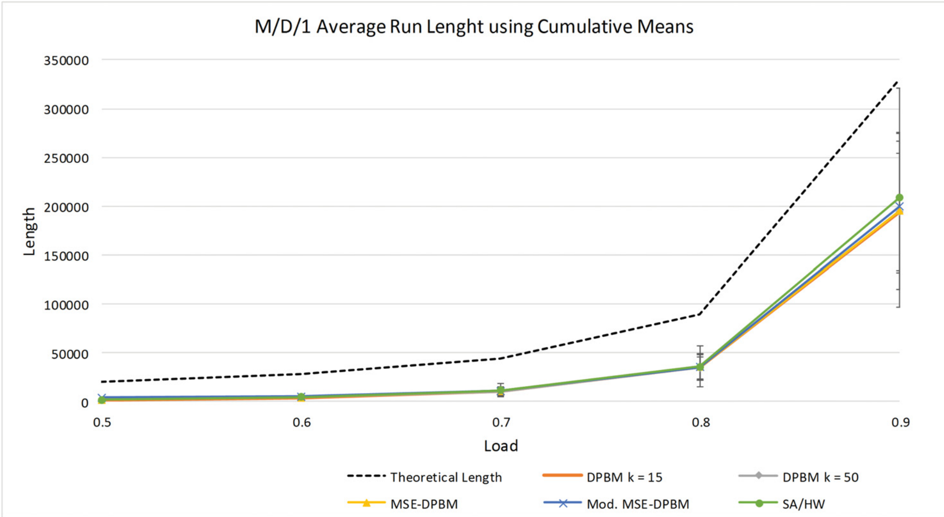


Figure 7.4: M/D/1's average run length per simulation using Cumulative Means

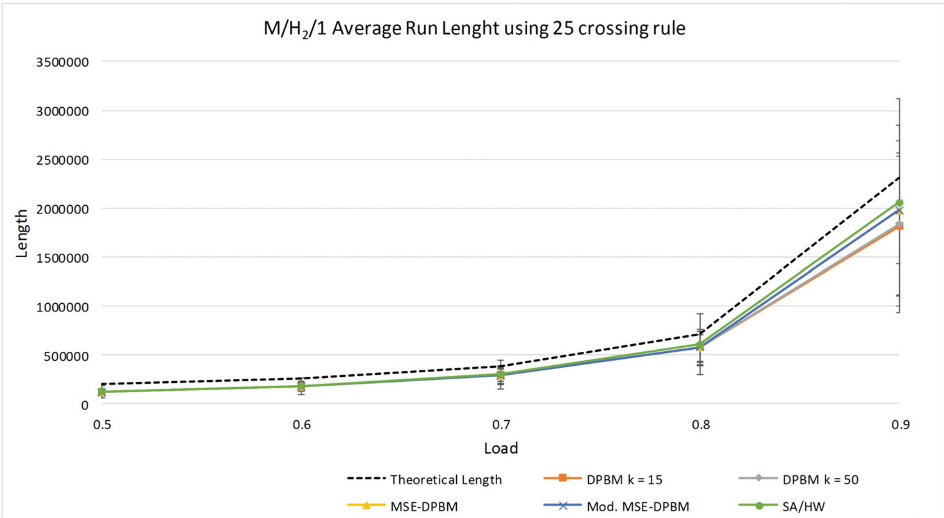


Figure 7.5: $M/H_2/1$'s average run length per simulation using 25 crossings rule

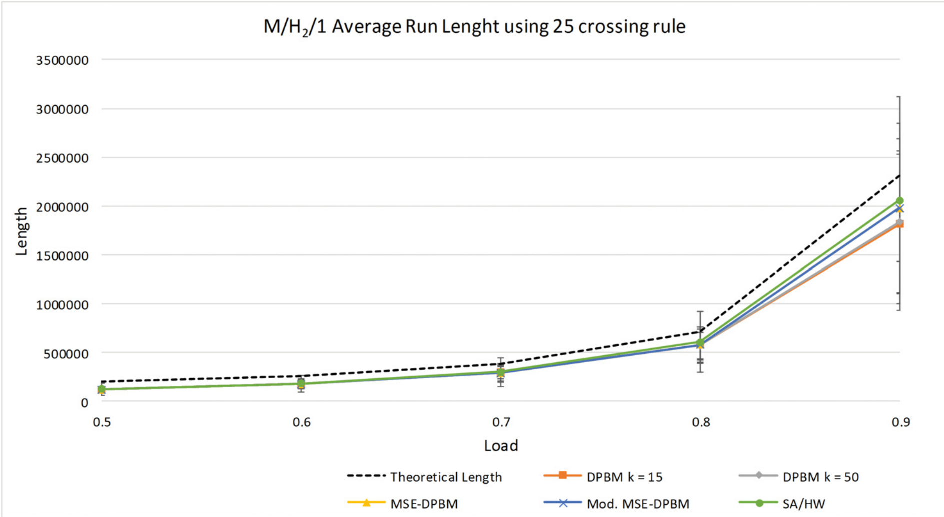


Figure 7.6: $M/H_2/1$'s average run length per simulation using Cumulative Means

From the results you can see that all the SOAMs are stopping, in average, the simulation too early. Therefore, we are expecting the experimental coverage to be lower than the preset theoretical level $1 - \alpha = 0.95$. On the other hand, the number of observations per simulation and model/load combination seems to be sufficiently close to the theoretical value and the DPBM variations can be used for the coverage analysis. It can be seen that SA/HW and Mod. MSE-DPBM are performing better in this experiment and especially while simulating $M/H_2/1$ model the difference is quite significant. The confidence intervals overlap for this experiment, therefore refer to Tables E.2, E.3 and E.4 for exact results if you are interested more in the actual values of $\sigma(\bar{L})$.

Comparing the 25 crossings rule and Cumulative Means method, we can see that they both do not affect the average simulation run length and are equivalent.

## 7.2 Experiment 2: Coverage Analysis

This section introduces the results of experimental coverage analysis, as set up in Section 5.7, for all of the SOAMs and models as described in Table 5.1. In all the graphs we have plotted the performance of DPBM variants and SA/HW over range of correlation coefficient $\phi = 0.5, 0.6, ..., 0.95$ for the AR(1) process and traffic intensity $\rho = 0.5, 0.6, ..., 0.95$ for the queueing processes. As error bars we have selected the absolute error $\Delta_{z-\frac{1}{2}}$, this is true unless otherwise stated.

### 7.2.1 AR(1)

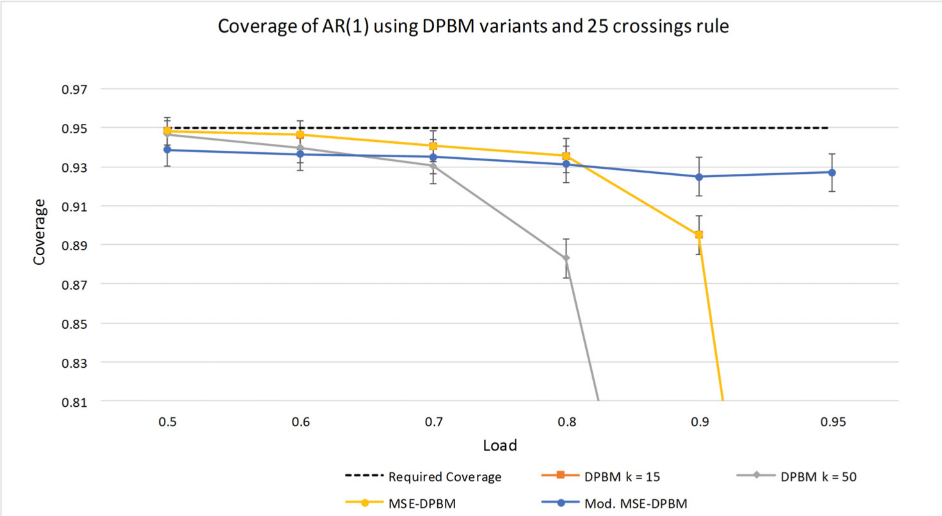


Figure 7.7: AR(1)'s coverage, using variants of DPBM and 25 crossings rule

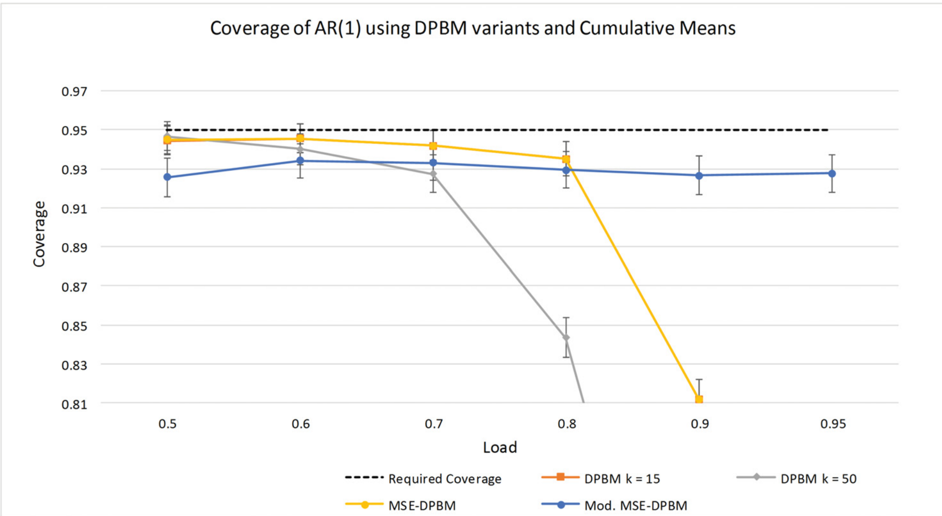


Figure 7.8: AR(1)'s coverage, using variants of DPBM and Cumulative Means

From the results in Figures 7.7 and 7.8 it can be seen that both DPBM methods and MSE-DPBM method perform very poorly for higher value of the correlation coefficient $\phi = 0.8, 0.9, 0.95$. The MSE-DPBM has very similar performance to DPBM ($k = 15$) and the two lines overlap. We assume that the problem is in the estimation of optimal batch size $\hat{m}^*$.

The Mod. MSE-DPBM performs better than other DPBM variants and is selected for comparison with SA/HW.

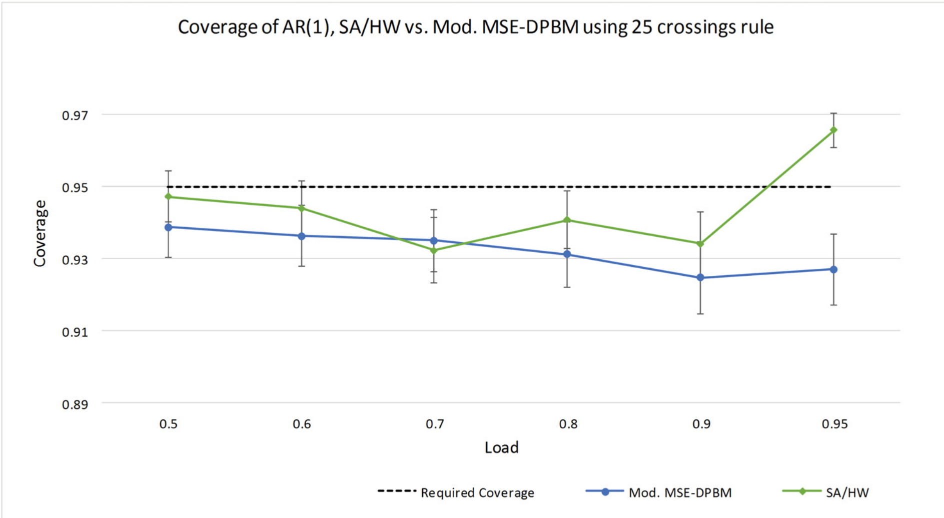


Figure 7.9: AR(1)'s coverage, SA/HW vs. Mod. MSE-DPBM using 25 crossings rule

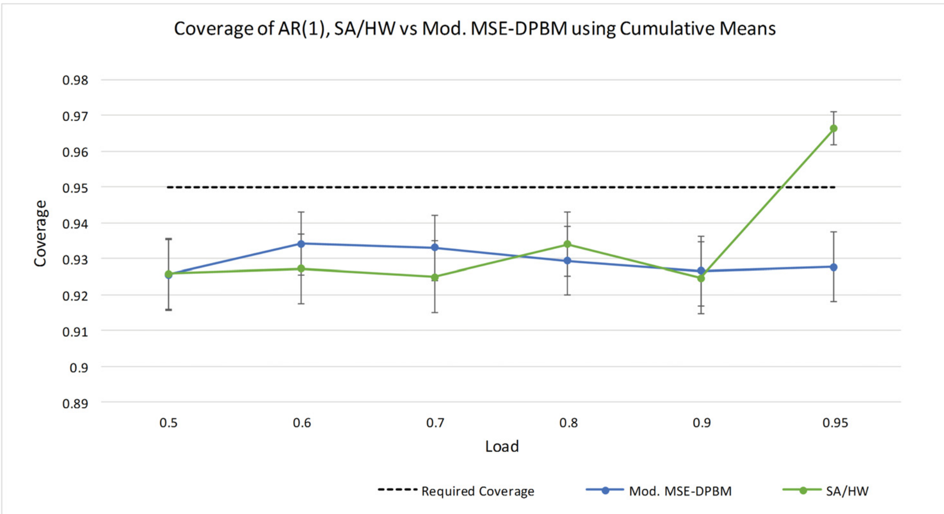


Figure 7.10: AR(1)'s coverage, SA/HW vs. Mod. MSE-DPBM using Cumulative Means

From the Figures 7.9 and 7.10 we conclude that SA/HW performs better for higher values of the autocorrelation coefficient $\phi$, where our main attention lies. The overestimation of CIs at $\phi = 0.95$ of SA/HW is lower than the underestimation of Mod. MSE-DPBM, therefore, we recommend using SA/HW based on the its quality of coverage by CI. On the other hand, both methods sufficiently good and usable for the analysis of AR(1).


Figure 7.11: AR(1)'s run length using 25 crossings rule

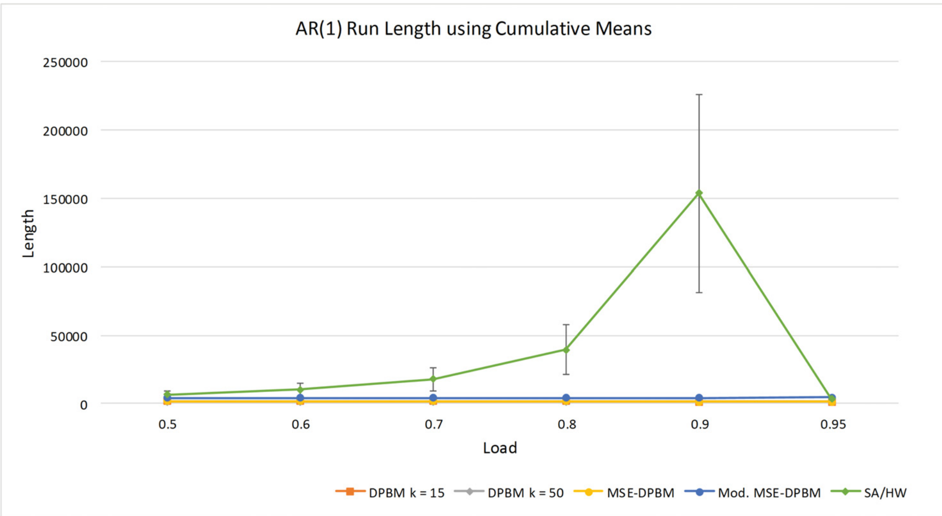


Figure 7.12: AR(1)'s run length using Cumulative Means

Comparing the average run-length from Figures 7.11 and 7.12 we can see that SA/HW and DPBM variants required, in average, similar number of observations per one simulation experiment of coverage analysis. However, we can see that SA/HW requires more observations at $\phi = 0.9$. All the DPBM variants require around 2000 to 4500 observations and SA/HW requires little bit over 150000. Please note that the error bars here are the standard deviations of the average length.

We don't see any significant difference between using 25 crossings rule and Cumulative Means methods of initial transition detection, however the it can be seen from Table E.1 that by using Cumulative Means required in average less observations than 25 crossings rule. For detailed results see Table E.1.

### 7.2.2 M/M/1

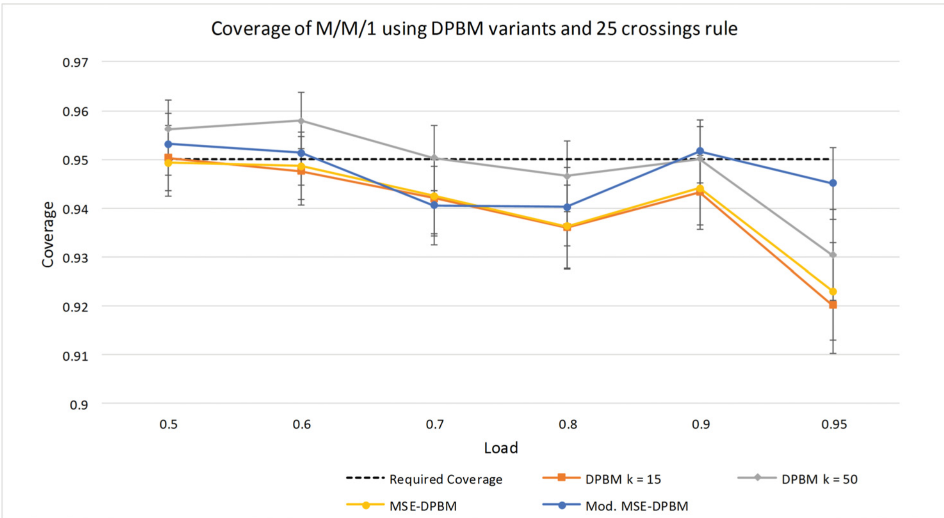


Figure 7.13: M/M/1's coverage, using variants of DPBM and 25 crossings rule

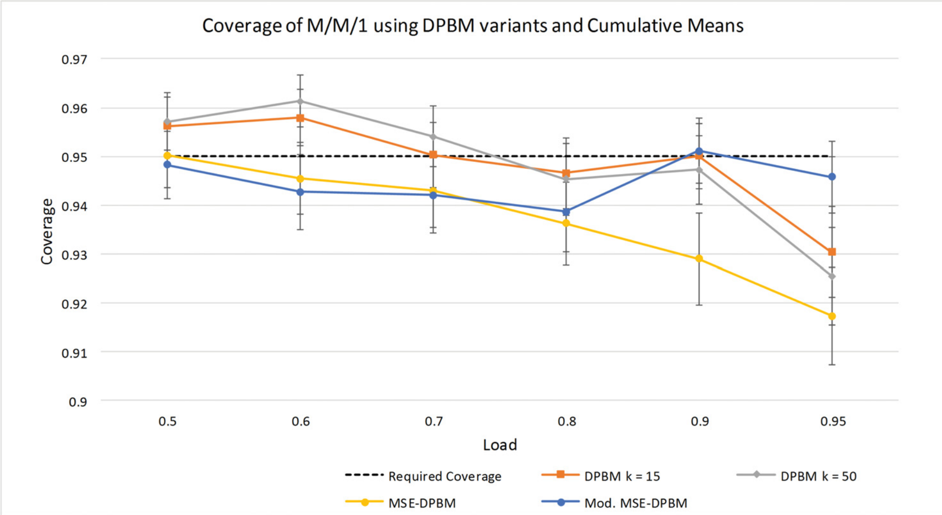


Figure 7.14: M/M/1's coverage, using variants of DPBM and Cumulative Means

From Figures 7.13 and 7.14 we can see that Mod. MSE-DPBM performs the best out of the DPBM variants. The method of DPBM $k = 50$ performs sufficiently well, however as mentioned in Section 6.2.1 the method cannot be used for as an automated component of Akaroa2. We select the Mod. MSE-DPBM to be compared with SA/HW as it performs the best and especially for higher traffic intensities $\rho = 0.9$ and 0.95 the performance is almost optimal.

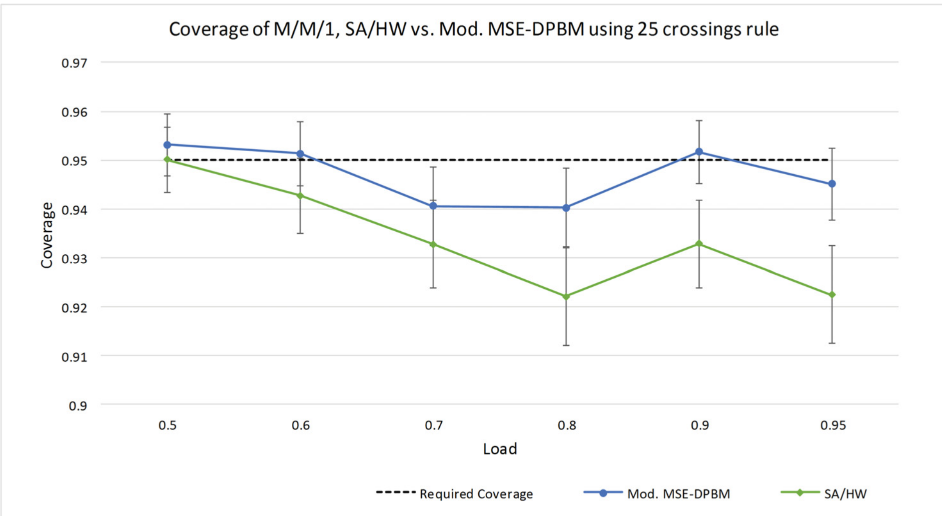


Figure 7.15: M/M/1's coverage, SA/HW vs. Mod. MSE-DPBM using 25 crossings rule

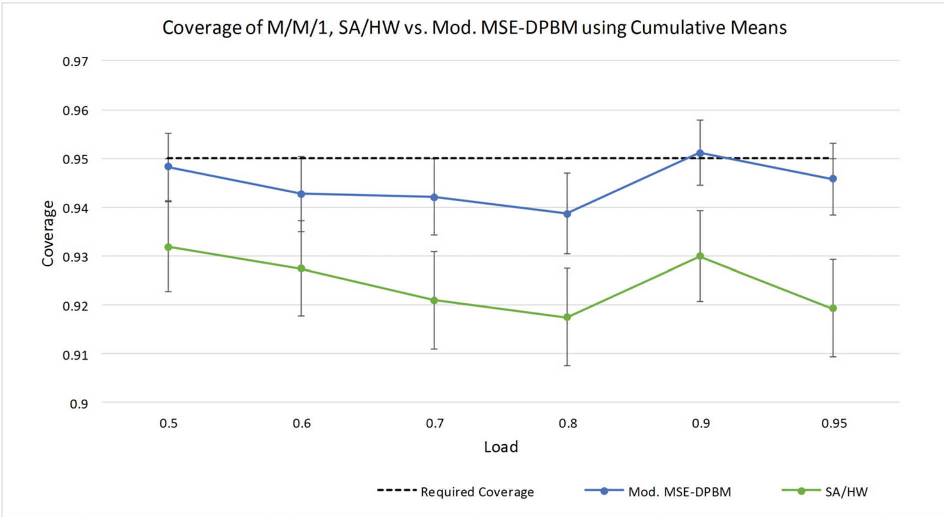


Figure 7.16: M/M/1's coverage, SA/HW vs. Mod. MSE-DPBM using Cumulative Means

From the Figures 7.15 and 7.16 we see that Mod. MSE-DPBM performs better for the whole range of traffic intensities $\rho = 0.5, ..., 0.95$. Run length per simulation was compared in Experiment 1, please see Section 7.1. However, method of SA/HW is still a suitable method.

Comparing the 25 crossings rule with Cumulative Means, we can see from Table E.2 that simulations with 25 crossings rule and SA/HW required longer independent simulations in the lower ranges of $\rho = 0.5, 0.6, 0.7$. The difference in average number of observations for DPBM variants is insignificant. However, using 25 crossings rule the coverage is closer to the required level of 0.95.

For detailed results see Table E.2.

### 7.2.3 M/D/1

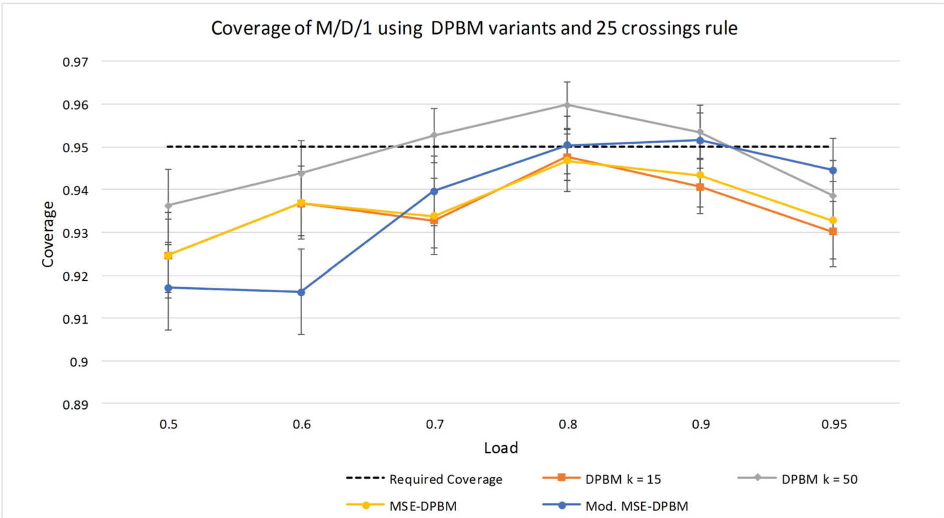


Figure 7.17: M/D/1's coverage, using variants of DPBM and 25 crossings rule

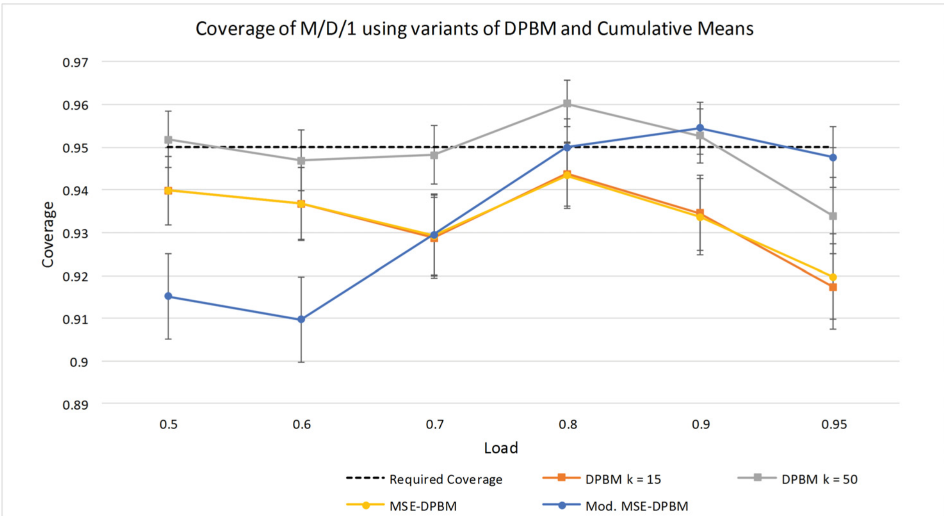


Figure 7.18: M/D/1's coverage, using variants of DPBM and Cumulative Means

From Figures 7.17 and 7.18 we can see that Mod. MSE-DPBM performs the best for higher loads of $\rho$ but poorly for lower loads of $\rho$. However, as our main focus lies in the higher loads, where the correlations are higher, we select the Mod. MSE-DPBM for the comparison with SA/HW. The DPBM

$k = 15$ performs very good in this scenario, however it cannot be used as an automated method under Akaroa2 6.2.1.

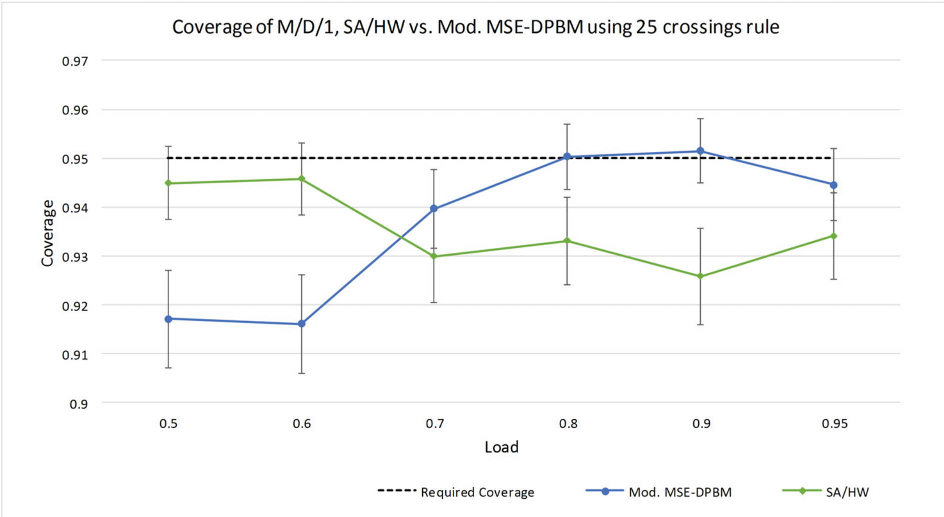


Figure 7.19: M/D/1's coverage, SA/HW vs. Mod. MSE-DPBM using 25 crossings rule

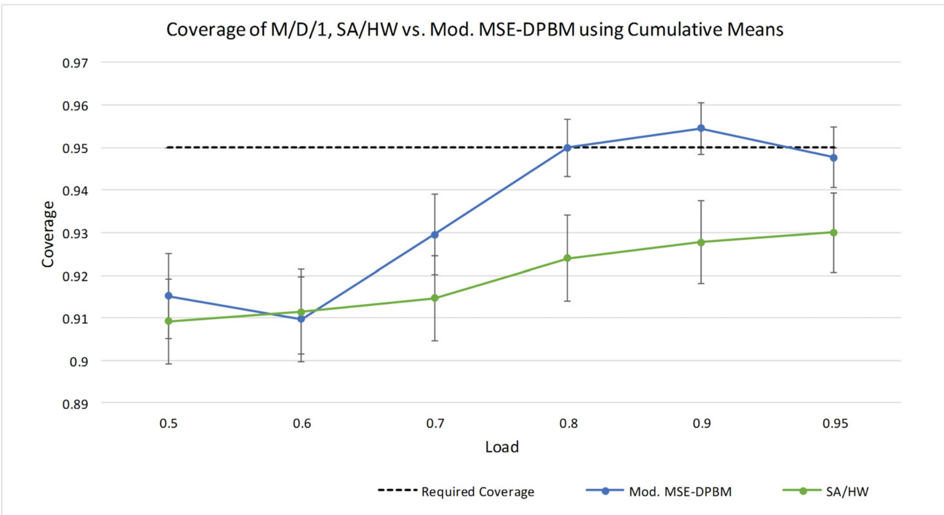


Figure 7.20: M/D/1's coverage, SA/HW vs. Mod. MSE-DPBM using Cumulative Means

Comparing the SA/HW and Mod. MSE-DPBM, see Figures 7.19 and 7.20, we can conclude that Mod. MSE-DPBM performs better overall than SA/HW. Both SA/HW and Mod. MSE-DPBM, while using Cumulative

Means, perform better for higher values of the traffic intensity $\rho$. Both methods perform sufficiently well for usage as an output analysis method of steady-state simulations. The average run lengths were compared in the Experiment 1 7.1.

Comparing the 25 crossings rule and Cumulative Means methods of initial transient detection we can see that both methods do not affect the quality of coverage highly, both methods here are, therefore, suitable.

### 7.2.4 M/H$_2$/1

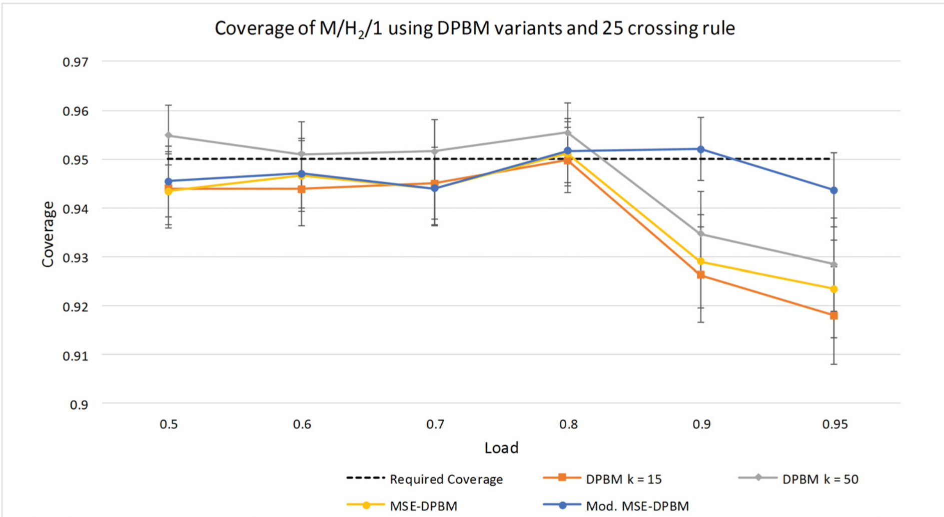


Figure 7.21: M/H$_2$/1's coverage, using variants of DPBM and 25 crossings rule

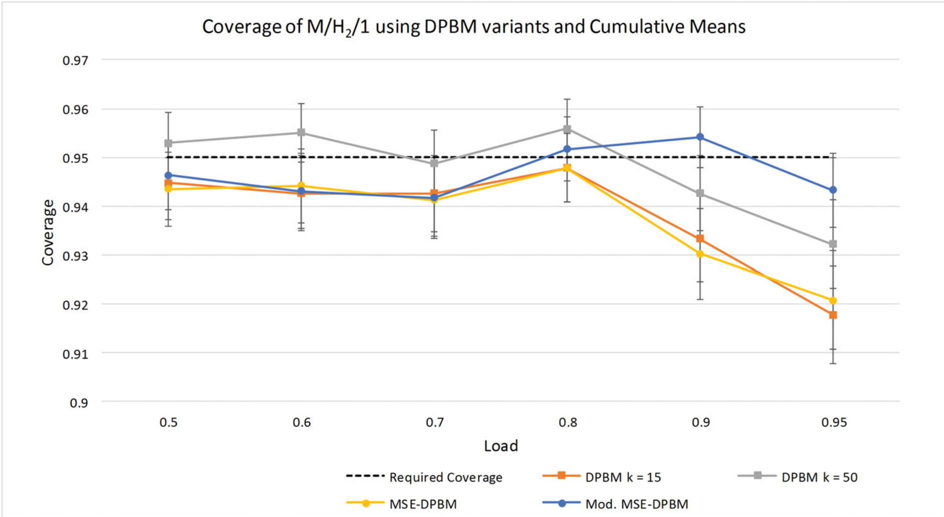


Figure 7.22: $M/H_2/1$'s coverage, using variants of DPBM and Cumulative Means

From the Figures 7.21 and 7.22 we can see that Mod. MSE-DPBM performs the best out of the DPBM variants. And is very close to the theoretical preset confidence level of $1 - \alpha = 0.95$ for all the range of traffic intensities $\rho$. Therefore we select the Mod. MSE-DPBM to compare with SA/HW. All the other DPBM variants under-perform for the higher loads, especially for $\rho = 0.9$ and 0.95. However, MSE-DPBM method here is usable. DPBM with $k = 15$ and $k = 50$ are sufficiently good, but cannot be used as an automated method.

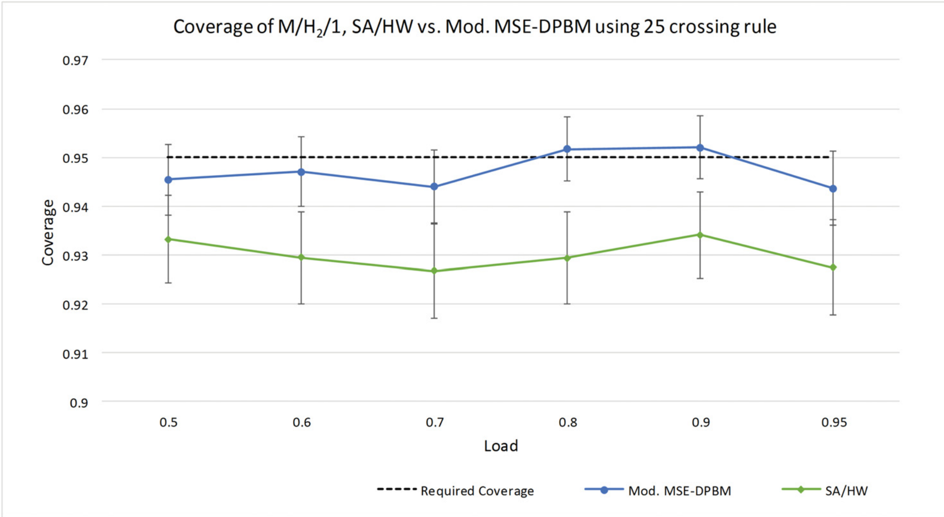


Figure 7.23: $M/H_2/1$'s coverage, SA/HW vs. Mod. MSE-DPBM using 25 crossings rule

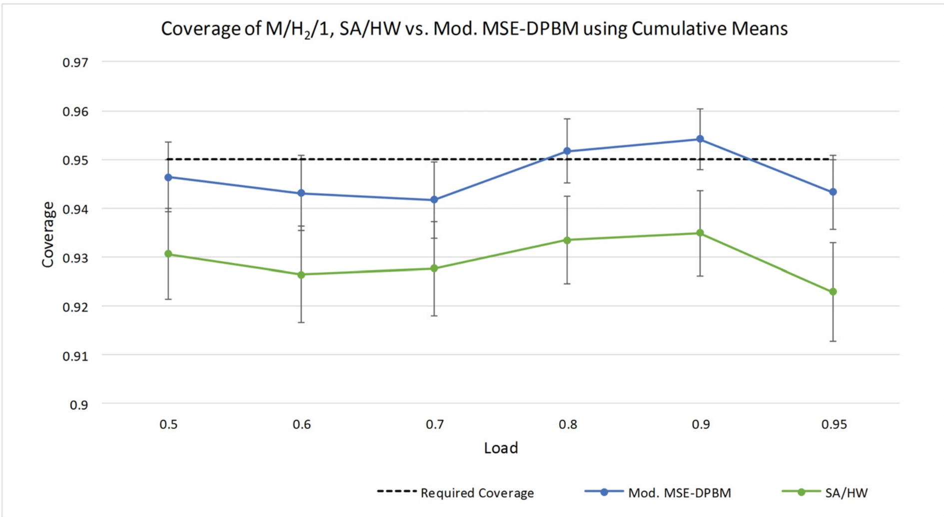


Figure 7.24: $M/H_2/1$'s coverage, SA/HW vs. Mod. MSE-DPBM using Cumulative Means

The Figure 7.23 and 7.24 show that Mod. MSE-DPBM performs better than SA/HW for the whole range of traffic intensities, where SA/HW approaches experimental coverage of 0.93 and 0.92 for 25 crossings rule and Cumulative means respectively. The Mod. MSE-DPBM keeps its experimental coverage very close to the value of 0.95. Here we can conluded that SA/HW and Mod. MSE-DPBM are both usable for simulation output analysis of stochastic steady-state simulation. The run lengths were compared in the Experiment 1 7.1.

Again we can see that while using the 25 crossings rule method of initial transient detection the experimental coverage is slightly better than while using Cumulative Means. For more details please refer to Table E.4.

### 7.2.5 Open Queueing Network

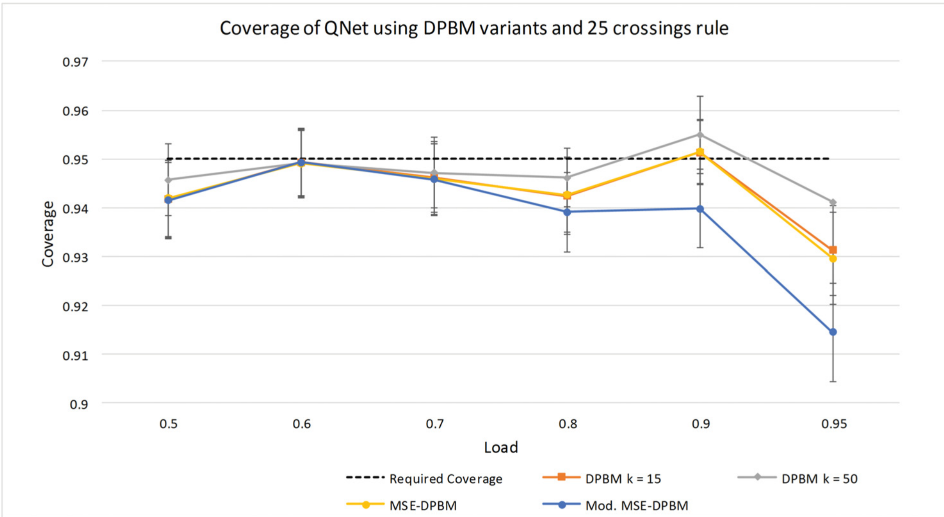


Figure 7.25: QNet's coverage, using variants of DPBM and 25 crossings rule

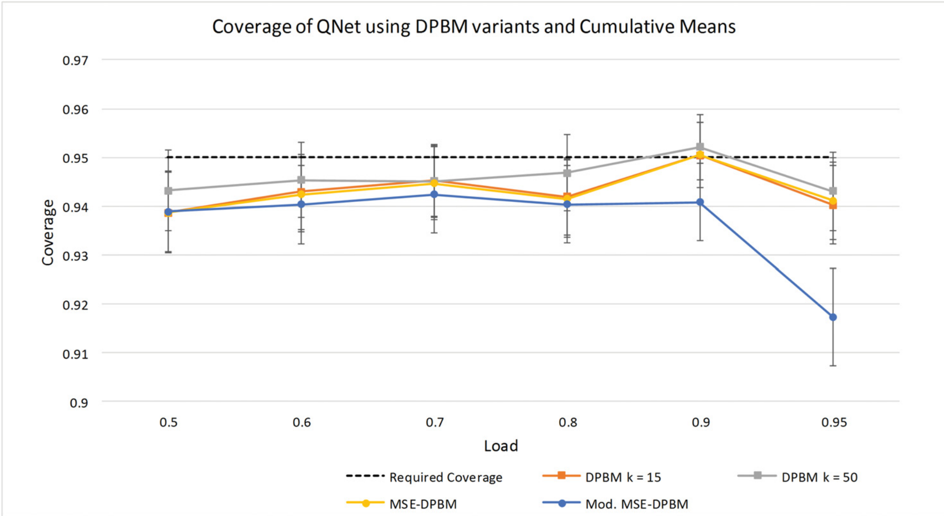


Figure 7.26: QNet's coverage, using variants of DPBM Cumulative Means

From the Figures 7.25 and 7.26 we can see that Mod. MSE-DPBM performs worse than all the other variants of DPBM. We, therefore, select the MSE-DPBM for the comparison to SA/HW. However, all the variants of DPBM perform sufficiently good, even Mod. MSE-DPBM's experimental coverage of $\tilde{0}.915$ for $\rho = 0.95$ is still acceptable.

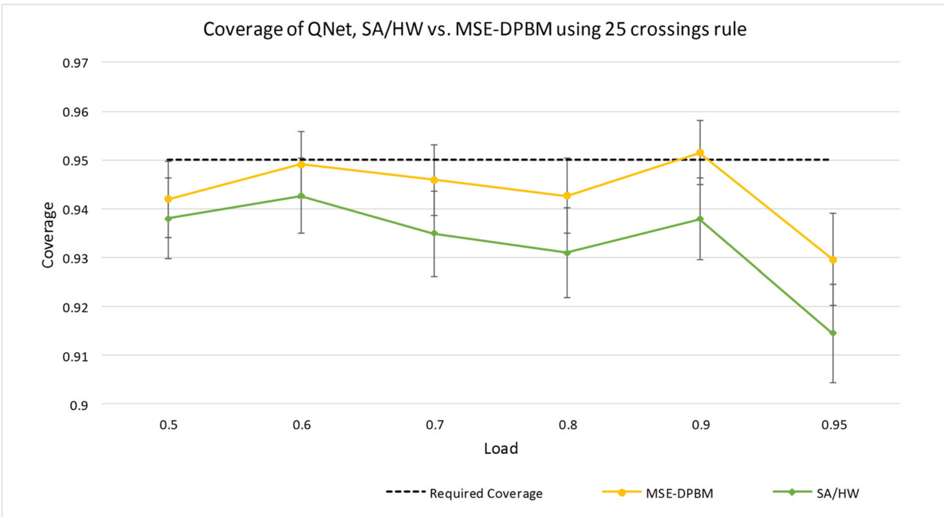


Figure 7.27: QNet's coverage, SA/HW vs. Mod. MSE-DPBM using 25 crossings rule

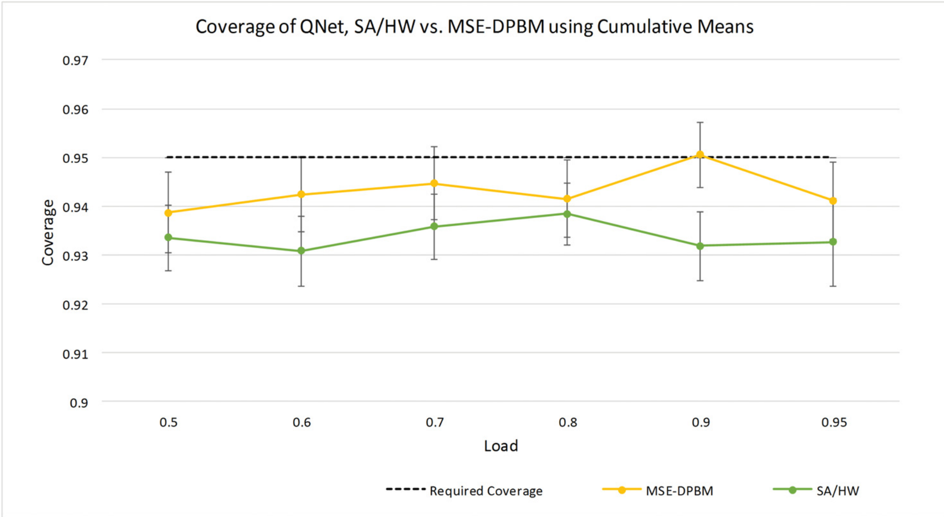


Figure 7.28: QNet's coverage, SA/HW vs. Mod. MSE-DPBM using Cumulative Means

The Figures 7.27 and 7.28 show that both methods of MSE-DPBM and SA/HW perform sufficiently good and can be used for stochastic steady-state simulations, where MSE-DPBM shows slightly better performance than SA/HW for our scenario.

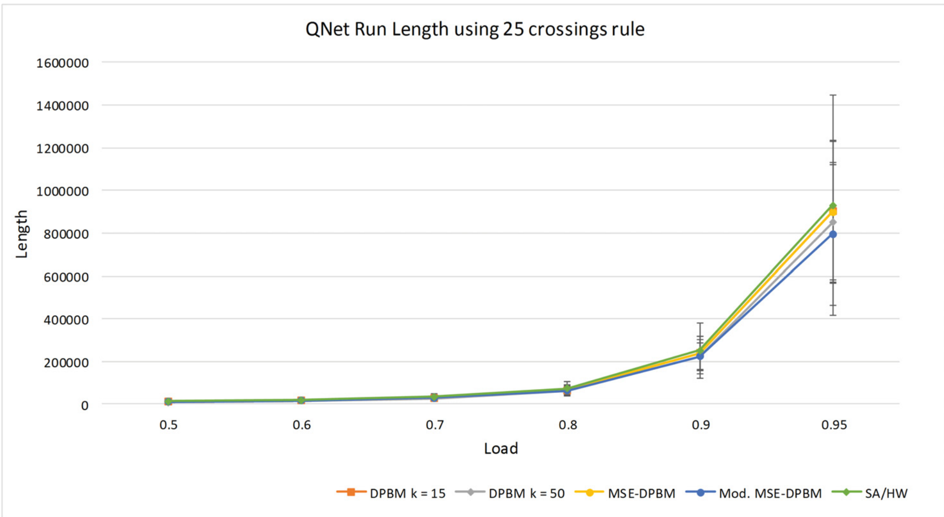


Figure 7.29: QNet Run length using 25 crossings rule

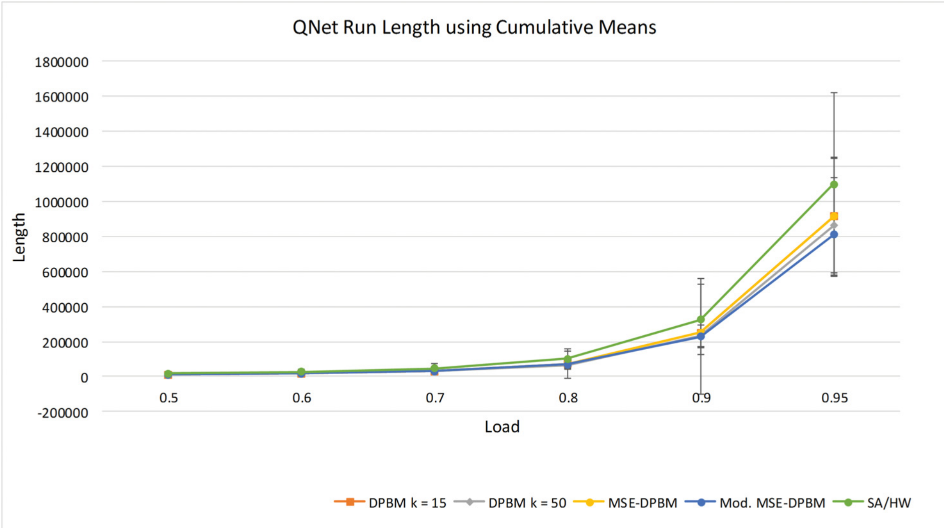


Figure 7.30: QNet Run length using Cumulative Means

The Figures 7.29 and 7.30 show that SA/HW and MSE-DPBM have longer average of number of observations per simulation. Which would make us think that these two methods would perform better in means of their coverage. This has shown true in the figures above.

Comparing the results from Figures 7.27 and 7.28 we can see that using Cumulative Means produces better experimental coverage. For more results please refer to Table E.5.

## 7.3 Experiment 3: Memory Requirements

In this experiment we show the memory requirements of MSE-DPBM, Mod. MSE-DPBM and SA/HW. We do not consider both DPBM versions for this experiment. The values are here for reference purposes only, as DPBM methods and SA/HW are not directly comparable. The methods based on batch means work better with more observations per batch as the observations $x_n$ in a batch become more independent and the independence is more important than a correlation between batches, see Section 3.2.3. SA/HW uses batches just to save memory as it stores only the sums of the observations $x_n$ 3.2.6. The checkpoint spacing for DPBM based methods is different than the one for SA/HW. The batch size $m$ and number of batches $k$ are shown here. The optimal batch size $\hat{m}^*$ is not shown in the graphs in order to keep them clean and will be discussed separately for each model, for results of $\hat{m}^*$ please refer to Table E.6. For all the models, the optimal batch size $\hat{m}^*$ performs in a similar manner, once the current batch size $m$ approaches the

$\hat{m}^*$ the $\hat{m}^*$ increases and until $m$ reaches to it again, the number of batches is increasing.

### 7.3.1 AR(1)

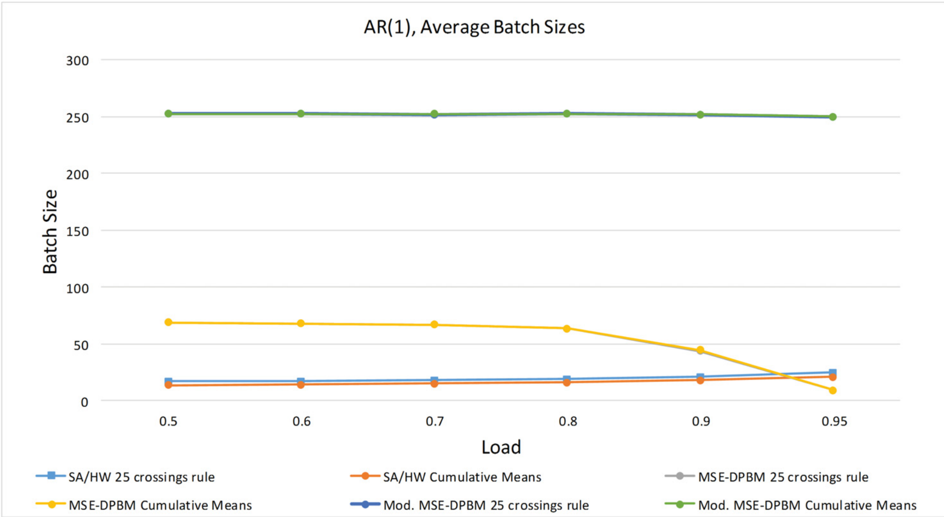


Figure 7.31: Average batch sizes recorded during the coverage experiment of AR(1) model

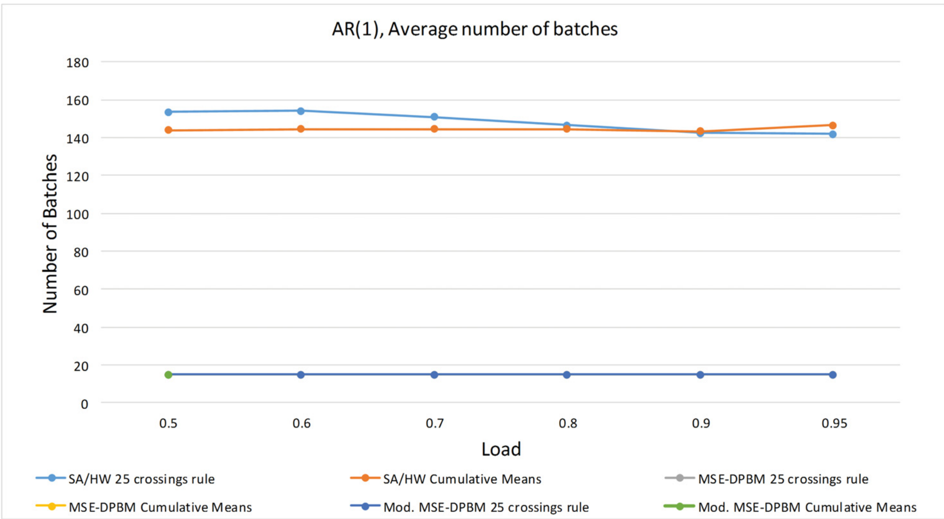


Figure 7.32: Average number of batches recorded during the coverage experiment

From the Figures 7.31 and 7.32 we see that SA/HW requires more and bigger batches in average than MSE-DPBM and Mod. MSE-DPBM. This is consistent with the results of coverage analysis, where SA/HW performed considerably better than DPBM methods, except the Mod. MSE-DPBM. Using method of Cumulative Means for the transient detection seems to require more observations per batch than 25 crossings rule, while keeping the number of batches essentially the same. From this experiment we conclude that using Mod. MSE-DPBM, out of the DPBM variants, with 25 crossings rule is the best option. SA/HW requires much bigger number of observations to produce essentially equal coverage as Mod. MSE-DPBM.

### 7.3.2 M/M/1

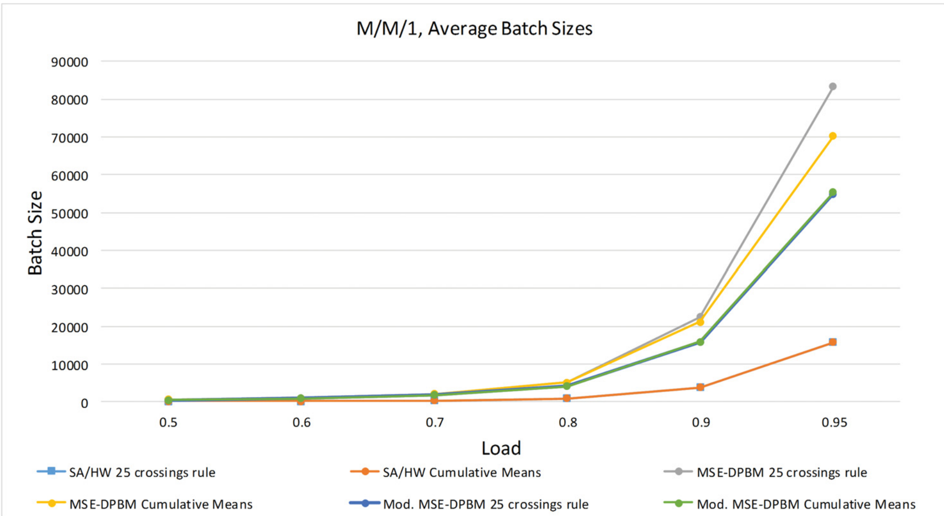


Figure 7.33: Average batch sizes recorded during the coverage experiment of M/M/1 model

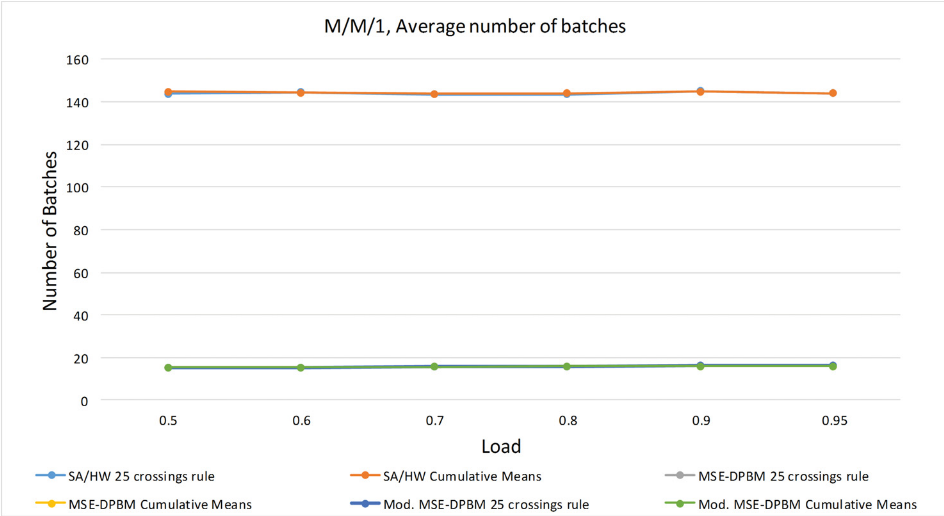


Figure 7.34: Average number of batches recorded during the coverage experiment of M/M/1 model

The Figures 7.33 and 7.34 show that Mod. MSE-DPBM requires lower amount of observations than MSE-DPBM, while keeping the number of batches essentially the same. This is consistent with the results of the coverage analysis experiment. However, using Cumulative Means here seems to require less observations to produce the final CIs. We recommend using either SA/HW with both methods of initial transient detection. Mod. MSE-DPBM with 25 crossings rule can be used instead of SA/HW, where SA/HW requires negligibly more observations, in average.

### 7.3.3 M/D/1

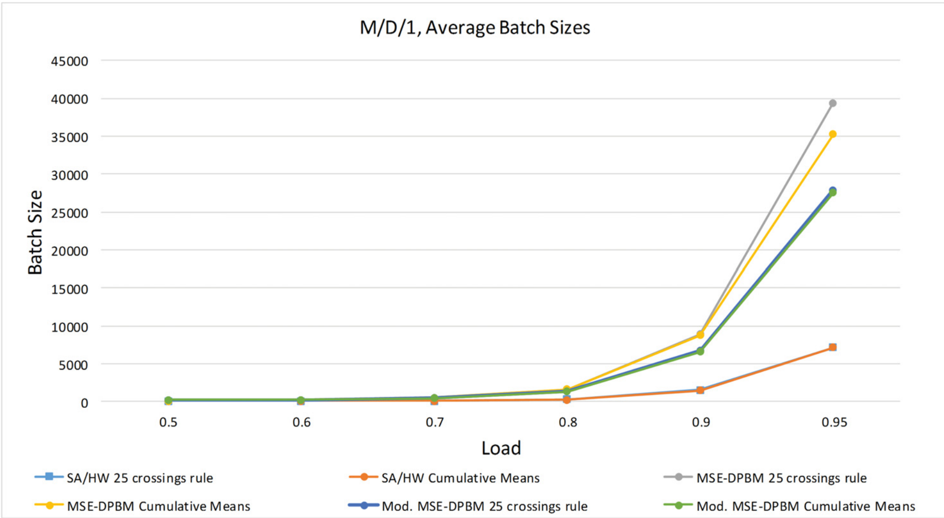


Figure 7.35: Average batch sizes recorded during the coverage experiment of M/D/1 model

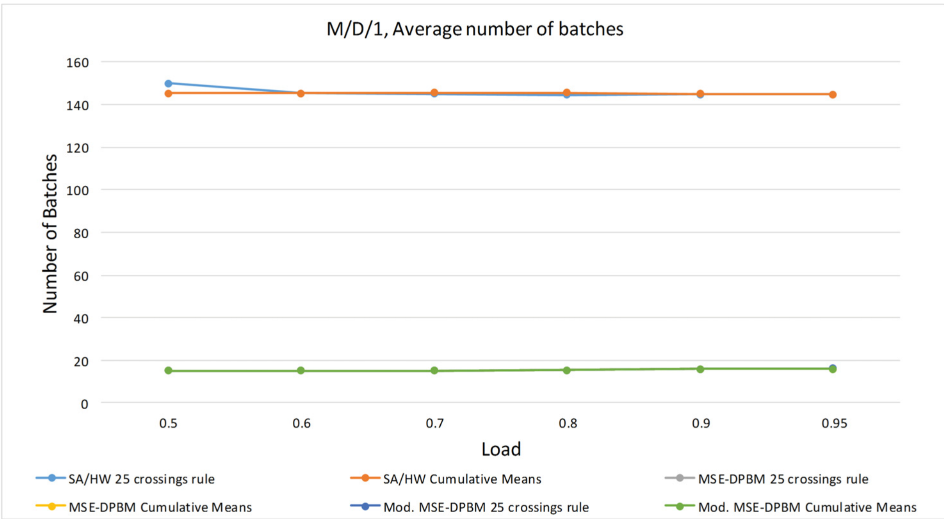


Figure 7.36: Average number of batches recorded during the coverage experiment of M/D/1 model

Figures 7.35 and 7.36 show same behaviour as the M/M/1 model.

### 7.3.4 M/H$_2$/1

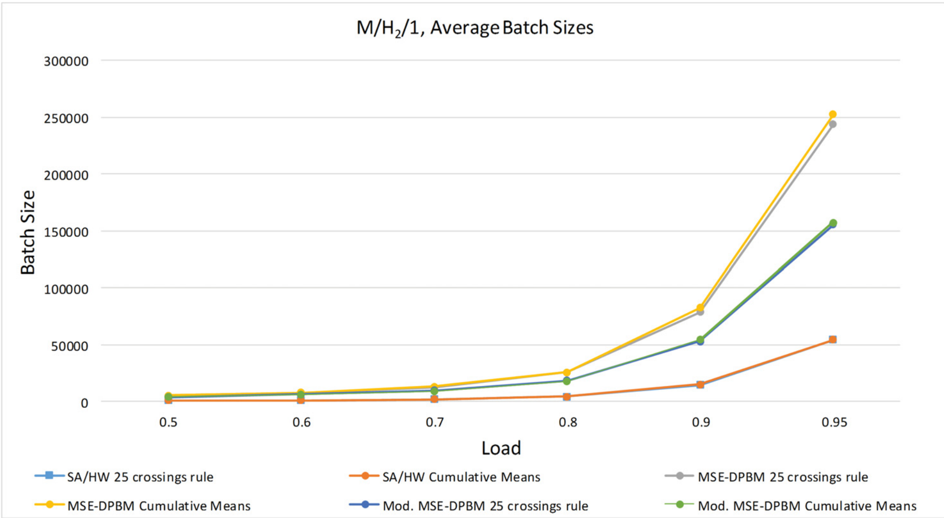


Figure 7.37: Average batch sizes recorded during the coverage experiment of M/H$_2$/1 model

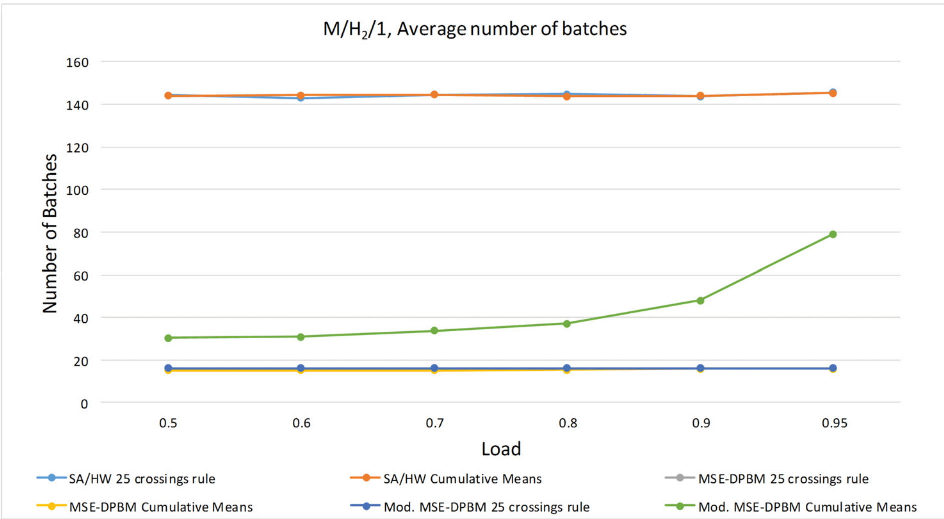


Figure 7.38: Average number of batches recorded during the coverage experiment of M/H$_2$/1 model

Figures 7.37 and 7.38 show same behaviour as the M/M/1 and M/D/1 model, except that Mod. MSE-DPBM with Cumulative Means requires on average more batches than all of the other DPBM implementations, while keeping the batch size equal to the one with using 25 crossings rule. We recommend using Mod. MSE-DPBM with 25 crossings rule or SA/HW as has been shown in coverage analysis experiment 7.2.

### 7.3.5 Queueing Network

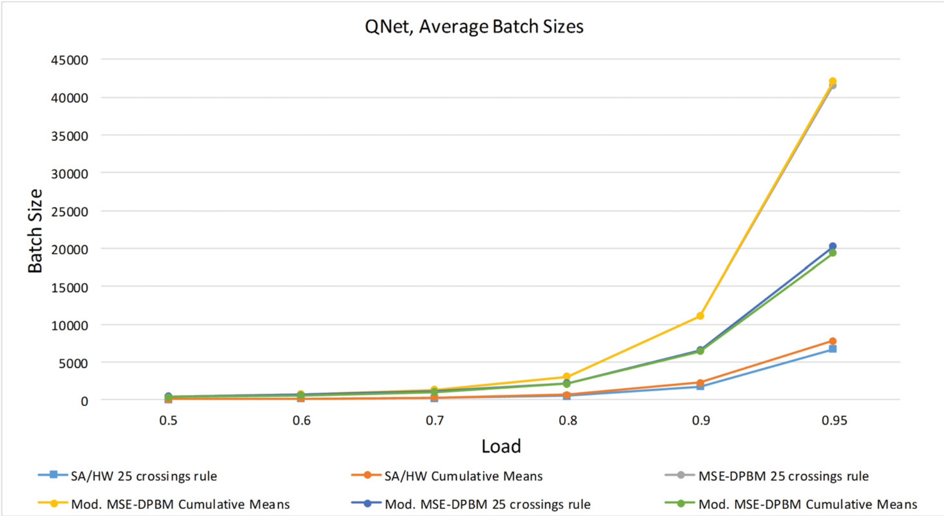


Figure 7.39: Average batch sizes recorded during the coverage experiment of queueing network model

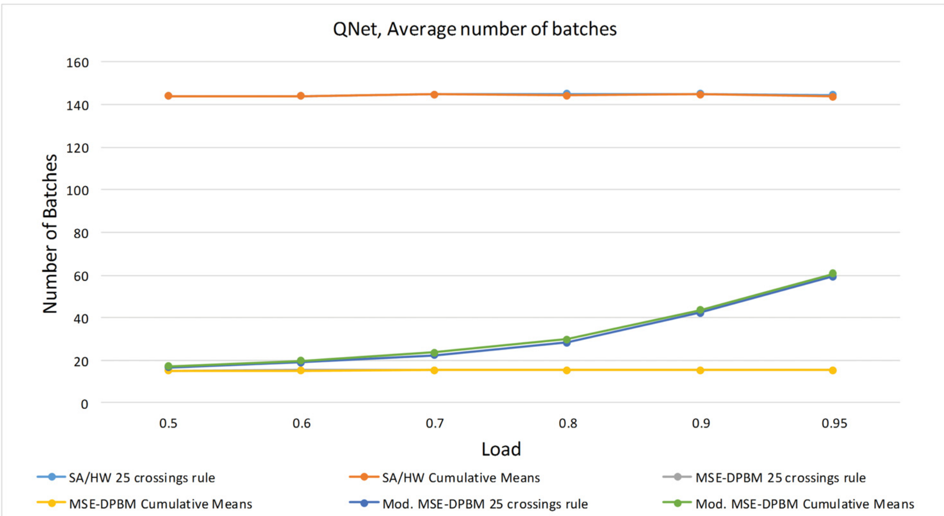


Figure 7.40: Average number of batches recorded during the coverage experiment of queueing network model

Figures 7.39 and 7.40 we can see that using Cumulative Means or 25 crossings rule as a method of initial transient detection is equally good. However, as we have seen in Experiment 2 7.2 MSE-DPBM performs the best in terms of coverage analysis, but from the results here we can see that it requires more observations than Mod. MSE-DPBM, therefore we recommend using the Mod. MSE-DPBM, as its experimental coverage is almost equal to the one of MSE-DPBM. As in the previous experiments, SA/HW is recommended to use, as well.

## 7.4 Average Number of Runs

In this experiment we show the number of runs that that are required to produce the final experimental coverage per each model. We compare the 25 crossings rule and Cumulative Means methods of initial transient detection, as well.

From the Figure 7.41 we can see that Mod. MSE-DPBM needed the least amount of runs to produce the final coverage results. Besides that SA/HW required equal number of runs up until $\phi = 0.95$, where it almost doubled the amount of Mod. MSE-DPBM. MSE-DPBM produced high number of runs, which did not cover the true mean in more than half cases 7.2. Using Cumulative Means reduces the number of runs highly, especially for the lower levels of $\phi$. At the higher level of $\phi, \phi = 0.9, 0.95$ the two methods produces essentially the same number of runs.

For models of M/M/1 (Figure 7.42), M/D/1 (Figure 7.43) and M/$H_2$/1 (Figure 7.44) the method of Cumulative Means shows better performance, as by using this method of initial transient we can see the coverage analysis required significantly less independent runs. SA/HW required less runs for the whole range of traffic intensities $\rho$ than both of the MSE-DPBM methods. Mod. MSE-DPBM required the most independent coverage experiments for all three models, however produced the best final coverage 7.2.

For the queueing network model, using the cumulative means method seem to require more independent runs for each method. The method of SA/HW here requires significantly more runs than the MSE-DPBM methods.

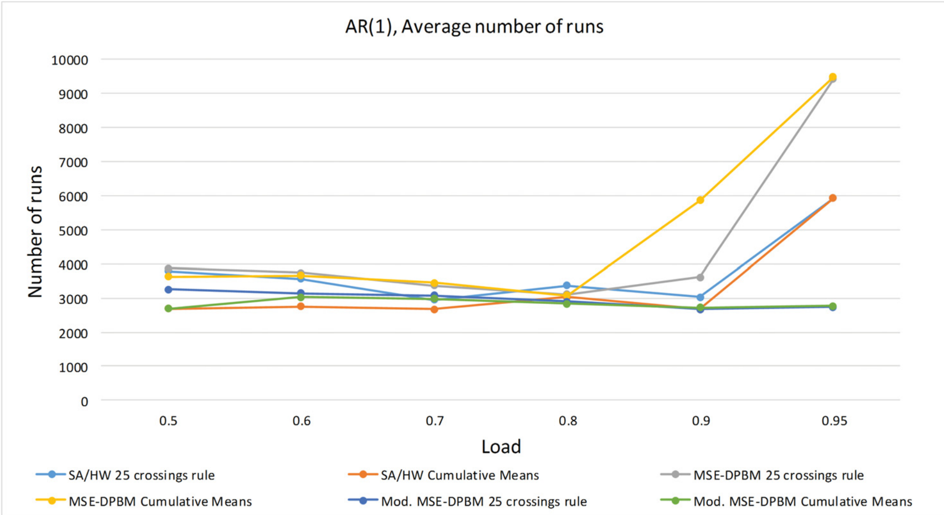


Figure 7.41: The number of runs required for the coverage experiment of AR(1)

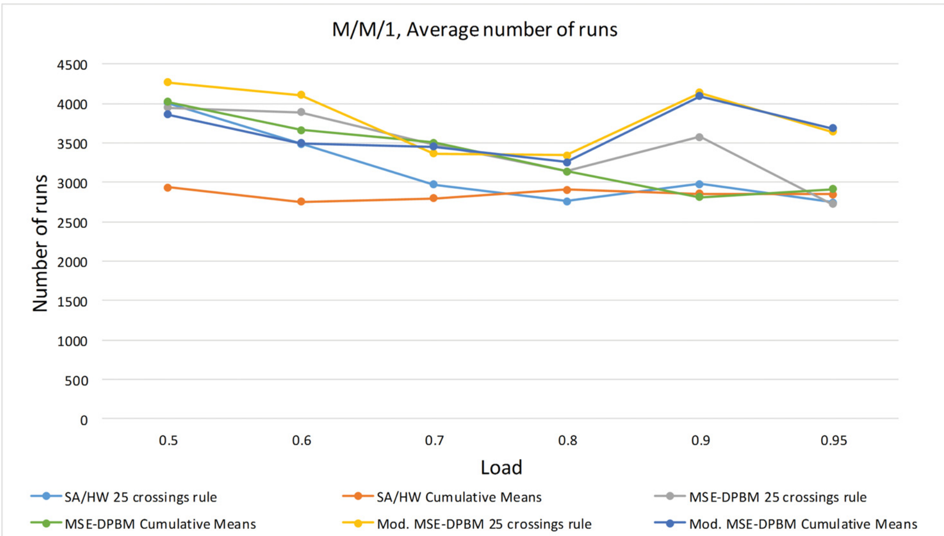


Figure 7.42: The number of runs required for the coverage experiment of M/M/1

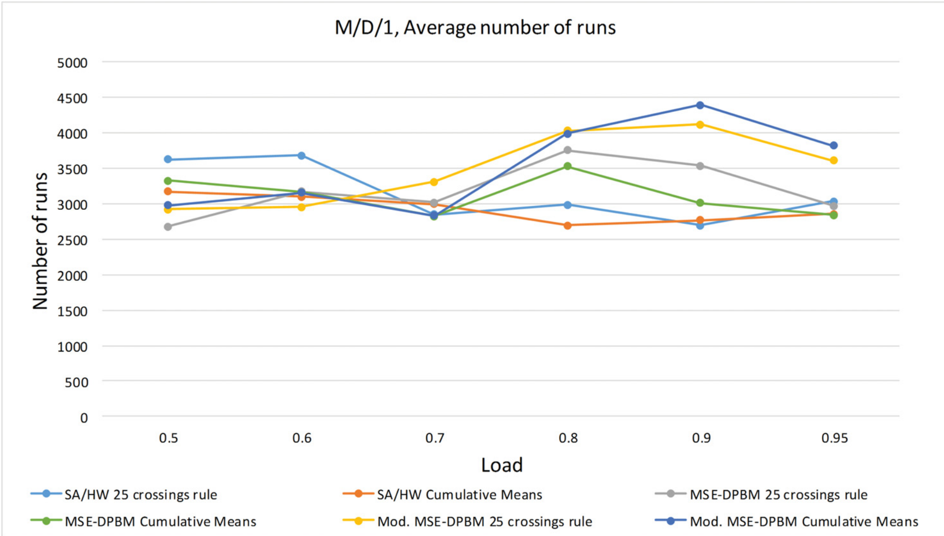


Figure 7.43: The number of runs required for the coverage experiment of M/D/1

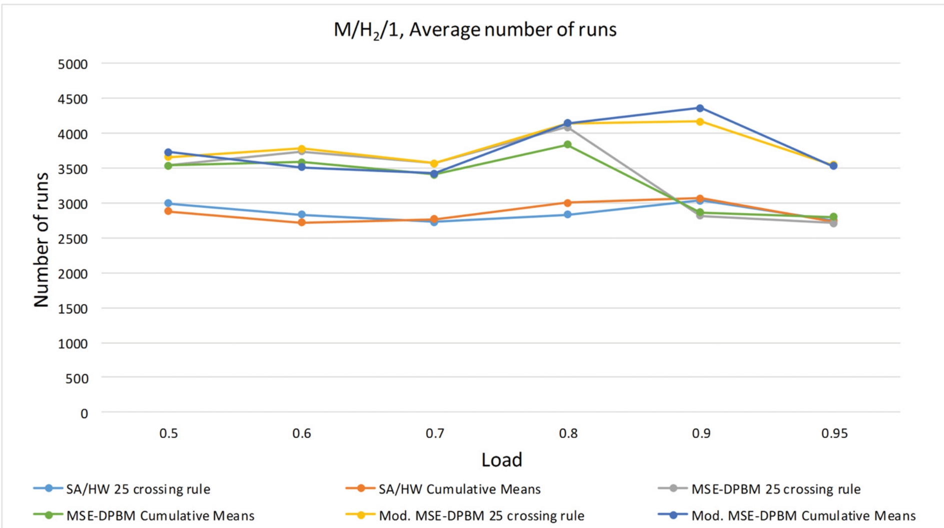


Figure 7.44: The number of runs required for the coverage experiment of $M/H_2/1$

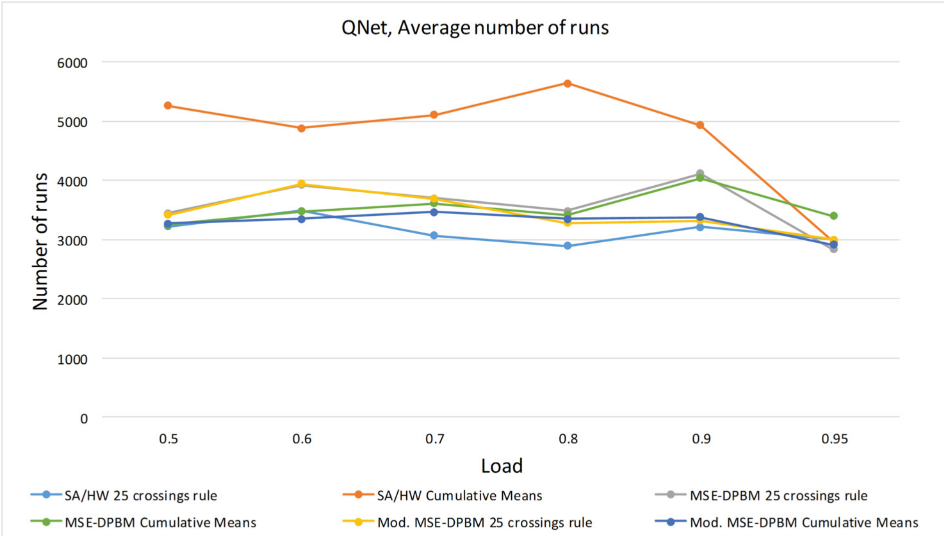


Figure 7.45: The number of runs required for the coverage experiment of queueing network

For more results accurate please refer to Tables E.1, E.2, E.3, E.4 and E.5

# Chapter 8

# Conclusions

From the research experiments we conclude that, first, the methods of DPBM and MSE-DPBM are implementable as components of Akaroa2 and can be, possibly, used as output analysis methods of sequential steady-state simulations. Secondly, the method of MSE-DPBM can be modified in order to perform better than MSE-DPBM in terms of experimental coverage analysis. The results of experimental coverage analysis show that DPBM and MSE-DPBM cannot be used in sequential stochastic steady-state simulations, as they perform badly for auto-regressive model and their coverage drops to 0.47 at $\phi = 0.95$ value of correlation coefficient of such process, however perform sufficiently well for the other 4 models (queueing models). On top of that, DPBM cannot be used as an automated method as its number of batches is fixed *a priory* of the simulation run. The Modified MSE-DPBM performs better than the other DPBM variants and also slightly better in terms of sequential experimental coverage analysis than SA/HW using a single processor scenario of MRIP. However, as it was shown in [22], introducing more simulations engines SA/HW improves in terms of coverage, where the batch means method does not. If we use multiple processor scenario of MRIP, as low as 4 independent replications of a simulation at one time, we inject an artificial independence for to SA/HW method, which by its nature works with correlated data and such independence improves its coverage quality. Where for batch means methods, which assume independence in their nature, introducing more simulation engines does not improve their performance.

We conclude that both methods, Mod. MSE-DPBM and SA/HW, are sufficiently good to be used for sequential stochastic steady-state simulations. But, we recommend to use the method of SA/HW as it requires, in average, less independent simulation runs to produce the final confidence intervals for coverage analysis. Therefore, is more stable method than Mod. MSE-DPBM, which produces more short “rejected” runs. We also compare the two methods of initial transient detection, namely 25 crossings rule and Cumulative Means. The results do not differ much and the results confirm the results found in [6] by Freeth. As we focus only on steady-state analysis of such processes we propose to use the method of Cumulative Means, as it gives much longer length of initial transient and is, therefore, safer to use for steady-state analysis. We also conclude that a method of detecting the length of the initial transient does not have much influence on the coverage or the run-length of a simulation. That also flows from the results shown by McNickle in [17], where it was shown that for longer simulations the truncation of observations does not have much influence on the final estimated values, only prevents the simulation of stopping too early.

From our results we can conclude that one should use SA/HW as implemented in Akaroa2 [16].

## 8.1 Answers to Research Questions

In this section we present the answers to our research questions, that have been asked in the Introduction section.

**1 - Are DPBM and MSE-DPBM implementable as a tool for steady-state simulation under Akaroa2?**

Yes, they are with few minor modifications. These include functions of Akaroa2 such as CheckpointReached(), GetCheckpoint(Checkpoint &cp) and ProcessObservation(real value). By including these methods and minor modifications to the algorithm of the methods the implementation as a component of Akaroa2 is possible. Checkpoint spacing has to be introduced to a artificial value. We have decided to send values to akmaster for analysis of stopping condition every time that full batch has been recorded after collapsing has occurred at least once.

For detailed results please refer to Sections 6.2.1 and 6.2.2.

**2 - Can MSE-DPBM be improved as a method of simulation output analysis?**

Yes, with few modifications. We have decided not to include the estimation of previous variance $\hat{V}_B(m/2)$ every time that estimation of optimal batch size $\hat{m}^*$ occurs. Rather we save current value of $\hat{V}_D PBM(m)$, that can be used later for the estimation as $\hat{V}_B(m/2)$ and computational time can be saved. Also we have included a waiting period, as the formulas for the estimation of optimal batch size assume that $n$ and $m$ are large enough,

to start estimation only after minimal number of observations has been collected, $m > 64$.
Please refer to Section 6.2.3 for details.

**3 - Which of the variants of DPBM perform the best in terms of coverage analysis?**

The Modified MSE-DPBM method performs better than the MSE-DPBM in 4 out of 5 experiment scenarios for coverage analysis. MSE-DPBM is better only at scenario simulating the open queueing network.

**4 - Are the DPBM variants accurate as an automated data analysis method in steady-state simulations?**

No, not all of them. The DPBM methods cannot be used as an automated methods, due to their fixed memory. MSE-DPBM is accurate for the queueing models, but performs badly for AR(1) model. Modified MSE-DPBM is accurate for all of our tested scenarios.
Please refer to Section 7.2.

**5 - Does a variant of DPBM perform better than Spectral Analysis in terms of coverage analysis?**

No. The Modified MSE-DBPM performs better for the single processor MRIP scenario, where SA/HW performs almost equally. By introducing the multiple processor scenario of MRIP we assume that the methods would be performing essentially equal. Mod. MSE-DPBM, however, shows less stability as more runs were rejected as "too short" than while using SA/HW.
Please refer to Section 7.2 and 7.3.

**6 - Is Schruben's test better than Cumulative Means as a method of initial transient detection?**

No, using the 25 crossings rule (please refer to Section 4.2.1 method of initial transient showed that, for 4 out of 5 (AR(1), M/M/1, M/D/1, M/$H_2$/1) of our reference models, it reduces the difference between theoretical confidence level $1 - \alpha$ and the experimental coverage. Using Cumulative Means, however, shows reduction in the amount of independent simulation runs for 4 out of 5 of models, especially for higher values of $\rho$ and $\phi$ where the correlations are strong.

However, as we focus on steady-state analysis we recommend using method of Cumulative Means as it shows better performance in the most complex of our simulated models, the open queueing network. Therefore, user can be safer that the initial transient has passed when the estimation phase starts.

## 8.2 Future Work

In future we would like to analyse the experimental coverage of a method developed by Alexopoulos et al. called "A sequential procedure for estimating the steady-state mean using standardized time series" [1]. The authors claim the methods is simpler than methods of batch means and can produce comparable quality of CIs.

Secondly, we would like to analyse why SA/HW required as many runs for $\phi = 0.90$ of AR(1) process, refer to Figures 7.11 and 7.11.

Thirdly, we would like perform coverage analysis under multiprocessor scenario of MRIP.

# Bibliography


[1] C. Alexopoulos, D. Goldsman, Peng Tang, and J.R. Wilson. A sequential procedure for estimating the steady-state mean using standardized time series. In *Simulation Conference (WSC), 2013 Winter*, pages 613–622, Dec 2013.

[2] Richard W Conway. Some tactical problems in digital simulation. *Management Science*, 10(1):47–61, 1963.

[3] G Ewing, D McNickle, and K Pawlikowski. Spectral analysis for confidence interval estimation under multiple replications in parallel. In *Proceedings of the 14th European Simulation Symposium, Dresden*, pages 52–61, 2002.

[4] G Ewing and K Pawlikowski. Akaroa ii version 2.4. 2 user's manual. Technical report, Department of Computer Science, University of Canterbury, 1997.

[5] Greg Ewing, Krzysztof Pawlikowski, and Don McNickle. *Akaroa-2: Exploiting network computing by distributing stochastic simulation*. SCSI Press, 1999.

[6] Adam Freeth. *Honours Report: A Sequential Steady-State Detection Method for Quantitative Discrete-Event Simulation*. Department of Computer Science and Software Engineering, University of Canterbury, Christchurch, New Zealand, 2012.

[7] James Douglas Hamilton. *Time series analysis*, volume 2. Princeton university press Princeton, 1994.

[8] Philip Heidelberger and Peter D Welch. Adaptive spectral methods for simulation output analysis. *IBM Journal of Research and Development*, 25(6):860–876, 1981.

[9] Philip Heidelberger and Peter D Welch. A spectral method for confidence interval generation and run length control in simulations. *Communications of the ACM*, 24(4):233–245, 1981.

[10] Philip Heidelberger and Peter D. Welch. Simulation run length control in the presence of an initial transient. *Operations Research*, 31(6):1109–1144, 1983.

[11] Peter Hellekalek. Good random number generators are (not so) easy to find. *Mathematics and Computers in Simulation*, 46(5):485–505, 1998.

[12] Leonard Kleinrock. *Queueing systems, volume I: Theory*. Wiley Interscience, 1975.

[13] Averill M. Law. *Simulation Modeling & Analysis 4th Edition*. McGraw-Hill Science, New York, 2006.

[14] Averill M. Law and W. David Kelton. Confidence intervals for steady-state simulations: I. a survey of fixed sample size procedures. *Operations Research*, 32(6):1221–1239, 1984.

[15] Pierre L'Ecuyer. Combined multiple recursive random number generators. *Operations Research*, 44(5):816–822, 1996.

[16] Don McNickle, Gregory Ewing, and Krzysztof Pawlikowski. Refining spectral analysis for confidence interval estimation in sequential simulation. *Proceedings of the ESS, Budapest*, pages 99–103, 2004.

[17] Don McNickle, Gregory C. Ewing, and Krzysztof Pawlikowski. Some effects of transient deletion on sequential steady-state simulation. *Simulation Modelling Practice and Theory*, 18(2):177–189, 2010.

[18] Don McNickle, Krzysztof Pawlikowski, and Greg Ewing. Akaroa2: A controller of discrete-event simulation which exploits the distributed computing resources of networks. In *Proceedings 24th European Conference on Modelling and Simulation (ECMS 2010)*, pages 104–109, 2010.

[19] Marc S Meketon and Bruce Schmeiser. Overlapping batch means: Something for nothing? In *Proceedings of the 16th conference on Winter simulation*, pages 226–230. IEEE Press, 1984.

[20] Krzysztof Pawlikowski. Steady-state simulation of queueing processes: survey of problems and solutions. *ACM Computing Surveys (CSUR)*, 22(2):123–170, 1990.

[21] Krzysztof Pawlikowski, H-DJ Jeong, and J-SR Lee. On credibility of simulation studies of telecommunication networks. *Communications Magazine, IEEE*, 40(1):132–139, 2002.

[22] Krzysztof Pawlikowski, Donald C McNickle, and Gregory Ewing. Coverage of confidence intervals in sequential steady-state simulation. *Simulation Practice and Theory*, 6(3):255–267, 1998.

[23] Krzysztof Pawlikowski, Victor WC Yau, and Don McNickle. Distributed stochastic discrete-event simulation in parallel time streams. In *Proceedings of the 26th conference on Winter simulation*, pages 723–730. Society for Computer Simulation International, 1994.

[24] Joseph William Schmidt and Robert Edward Taylor. *Simulation and analysis of industrial systems*. RD Irwin, 1970.

[25] Marcus Schoo, Krzysztof Pawlikowski, and Donald C McNickle. A survey and empirical comparison of modern pseudo-random number generators for distributed stochastic simulations. Technical report, Department of Computer Science and Software Development, University of Canterbury, 2005.

[26] Wheyming T Song and Mingchang Chih. Extended dynamic partial-overlapping batch means estimators for steady-state simulations. *European Journal of Operational Research*, 203(3):640–651, 2010.

[27] Wheyming Tina Song. A finite-memory algorithm for estimating the variance of the sample mean. *IIE Transactions*, 39(7):703–711, 2007.

[28] Wheyming Tina Song and Mingchang Chih. Run length not required: Optimal-mse dynamic batch means estimators for steady-state simulations. *European Journal of Operational Research*, 229(1):114–123, 2013.

[29] Peter D Welch. On the relationship between batch means, overlapping means and spectral estimation. In *Proceedings of the 19th conference on Winter simulation*, pages 320–323. ACM, 1987.

[30] Yingchieh Yeh and Bruce Schmeiser. Simulation output analysis via dynamic batch means. In *Proceedings of the 32nd conference on Winter simulation*, pages 637–645. Society for Computer Simulation International, 2000.

# Appendices

# Appendix A

# DPBM

```
/* Modification of DPBM as implemented in Song 2010 */

#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <vector>
#include <cmath>

//include the header file
#include "dpbm_variance_estimator.H"
#include "environment.H"
#include "checkpoint.H"
#include "akaroa/ak_message.H"

int checkpointPointer = 0;

DefineVarianceEstimatorType("DPBM", DynamicBMVarianceEstimator)

DynamicBMVarianceEstimator::DynamicBMVarianceEstimator(Environment *env, long trans){
    InitializeMethodVariables();
    InitializeMethodVectors();
}

DynamicBMVarianceEstimator::~DynamicBMVarianceEstimator(){
  //destructor
}

void DynamicBMVarianceEstimator::
ProcessObservation(real value)
{
    StartProcessingObservation(value);
}

boolean DynamicBMVarianceEstimator::ReachedCheckpoint(){
    if((checkpointPointer == 1)  && (m1 == batchSize)){
        ComputeEstimatorDPBM();
```

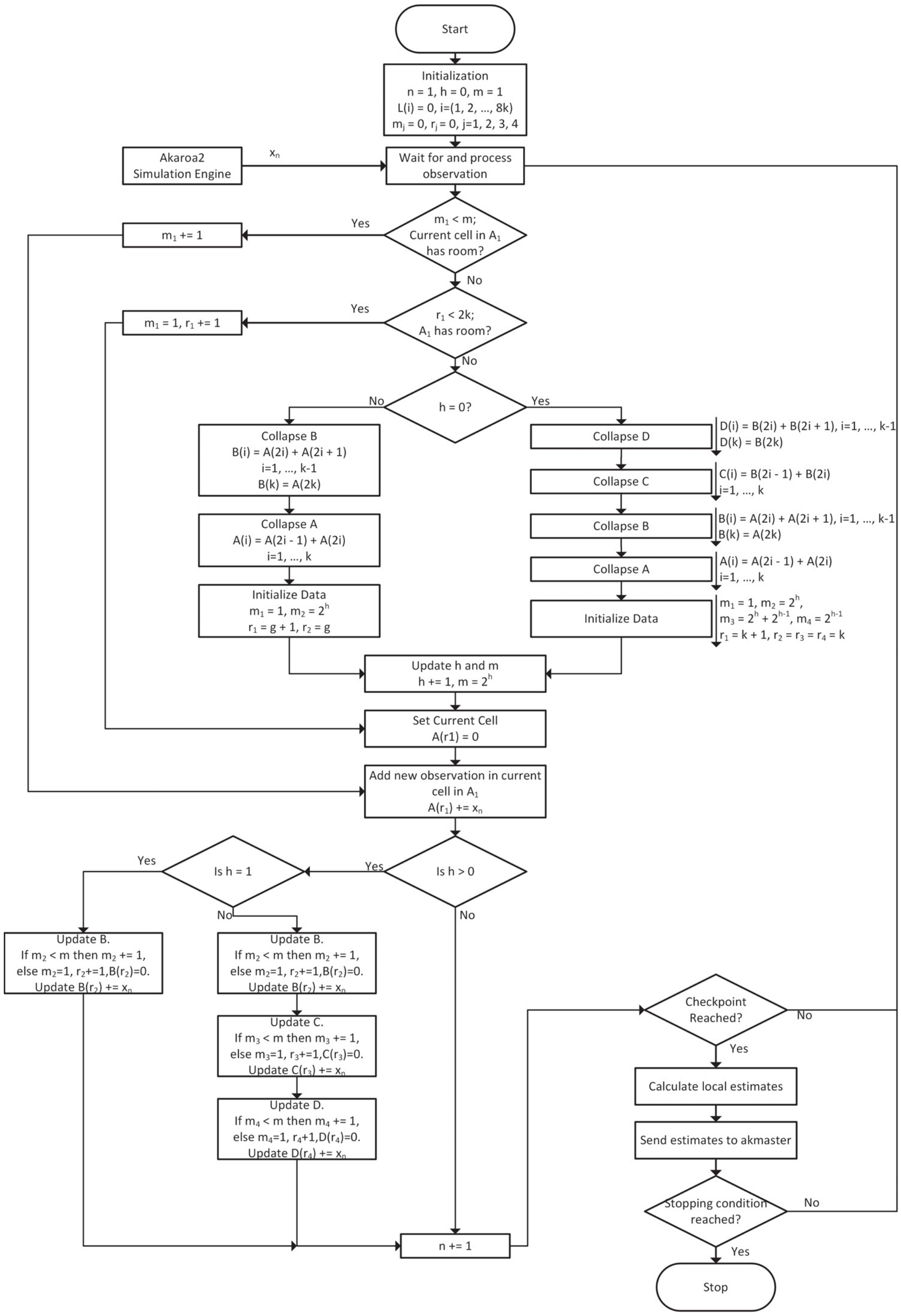


Figure A.1: DPBM Algorithm as implemented in Akaroa2

```
        return true;
    } else {
```

```
        return false;
    }

}

boolean DynamicBMVarianceEstimator::GetCheckpoint(Checkpoint &cp){
    cp.df = 0;
    cp.mean = mean;
    cp.variance = estimatorDPBM;
    return true;
}

void DynamicBMVarianceEstimator::InitializeMethodVariables(){
    currentSampleSizeN = 1;
    numberOfCollapsesOccured = 0;
    batchSize = pow(2, numberOfCollapsesOccured);
    m1 = 0, m2 = 0, m3 = 0, m4 = 0;
    r1 = 0, r2 = 0, r3 = 0, r4 = 0;
    memorySize = 15;
    return;
}

void DynamicBMVarianceEstimator::InitializeMethodVectors(){
    for(int i = 0; i < (2*memorySize); i++){
        observationVectorA.push_back(0.00);
        observationVectorB.push_back(0.00);
        observationVectorC.push_back(0.00);
        observationVectorD.push_back(0.00);
    }
    return;
}

void DynamicBMVarianceEstimator::StartProcessingObservation(real value){
    if(m1 < batchSize){
        m1 += 1;
        AddObservation(value);
        EndObservation(value);
        return;
    } else {
        m1 = 1;
        VectorHasRoom(value);
        return;
    }
}

void DynamicBMVarianceEstimator::VectorHasRoom(real value){
    if(r1 < ((2*memorySize)-1)){
        IncreasePointer();
        SetCurrentCell();
        AddObservation(value);
        EndObservation(value);
        return;
    } else {
        if(numberOfCollapsesOccured == 0){
```

```
            CollapseVectorB();
            CollapseVectorA();
            InitializeData();
            UpdateKandM();
            SetCurrentCell();
            AddObservation(value);
            EndObservation(value);
            return;
        } else {
            CollapseVectorD();
            CollapseVectorC();
            CollapseVectorB();
            CollapseVectorA();
            checkpointPointer = 1;
            InitializeData();
            UpdateKandM();
            SetCurrentCell();
            AddObservation(value);
            EndObservation(value);
            return;
        }
    }
}

void DynamicBMVarianceEstimator::InitializeData(){
    if(numberOfCollapsesOccured == 0){
        m1 = 1;
        m2 = pow(2, numberOfCollapsesOccured);
        r1 = memorySize;
        r2 = memorySize - 1;
        return;
    } else {
        m1 = 1;
        m2 = pow(2, numberOfCollapsesOccured);
        m3 = pow(2, numberOfCollapsesOccured) + pow(2, (
            numberOfCollapsesOccured -1));
        m4 = pow(2, (numberOfCollapsesOccured - 1));
        r1 = memorySize;
        r2 = memorySize - 1;
        r3 = memorySize - 1;
        r4 = memorySize - 1;
        return;
    }
}

void DynamicBMVarianceEstimator::UpdateKandM(){
    numberOfCollapsesOccured += 1;
    batchSize = pow(2,numberOfCollapsesOccured);
    return;
}

void DynamicBMVarianceEstimator::SetCurrentCell(){
    observationVectorA[r1] = 0;
    return;
```

```
}

void DynamicBMVarianceEstimator::CollapseVectorD(){
    for(int i = 1; i <= (memorySize-1); i++){
        observationVectorD[i-1] = observationVectorB[(2*i)-1] +
            observationVectorB[2*i];
    }
    observationVectorD[memorySize-1] = observationVectorB[(2*memorySize)-1];
    m4 = m2;
    return;
}

void DynamicBMVarianceEstimator::CollapseVectorC(){
    for(int i = 1; i <= (memorySize);i++){
        observationVectorC[i-1] = (observationVectorB[(2*i)-2] +
            observationVectorB[(2*i)-1]);
    }
    m3 = m2 * pow(2, numberOfCollapsesOccured);
    return;
}

void DynamicBMVarianceEstimator::CollapseVectorB(){
    for(int i = 1; i <= (memorySize-1); i++){
        observationVectorB[i-1] = observationVectorA[(2*i)-1] +
            observationVectorA[2*i];
    }
    observationVectorB[memorySize-1] = observationVectorA[(2*memorySize)-1];
    return;
}

void DynamicBMVarianceEstimator::CollapseVectorA(){
    for(int i = 1; i <= (memorySize);i++){
        observationVectorA[i-1] = (observationVectorA[(2*i)-2] +
            observationVectorA[(2*i)-1]);
    }
    return;
}

void DynamicBMVarianceEstimator::IncreasePointer(){
    m1 = 1;
    r1 += 1;
    return;
}

void DynamicBMVarianceEstimator::EndObservation(real value){
    if(numberOfCollapsesOccured == 1 && numberOfCollapsesOccured > 0){
        UpdateB(value);
    }else if(numberOfCollapsesOccured > 1){
        UpdateD(value);
        UpdateC(value);
        UpdateB(value);
    }
    currentSampleSizeN += 1;
    return;
```

```
}

void DynamicBMVarianceEstimator::UpdateB(real value){
    if(m2 < batchSize){
        m2 += 1;
    } else {
        m2 = 1;
        r2 += 1;
        observationVectorB[r2] = 0;
    }
    observationVectorB[r2] += value;
    return;
}

void DynamicBMVarianceEstimator::UpdateC(real value){
    if(m3 < batchSize){
        m3 += 1;
    } else {
        m3 = 1;
        r3 += 1;
        observationVectorC[r3] = 0;
    }
    observationVectorC[r3] += value;
    return;
}

void DynamicBMVarianceEstimator::UpdateD(real value){
    if(m4 < batchSize){
        m4 += 1;
    } else {
        m4 = 1;
        r4 += 1;
        observationVectorD[r4] = 0;
    }
    observationVectorD[r4] += value;
    return;
}

void DynamicBMVarianceEstimator::AddObservation(real value){
    observationVectorA[r1] += value;
    return;
}

void DynamicBMVarianceEstimator::ComputeMean(int b1){
    mean = 0;
    for(int i = 0; i < b1; i++){
        mean += (observationVectorA[i]);
    }
    mean = (mean/double(currentSampleSizeN));
    return;
}

void DynamicBMVarianceEstimator::ComputeEstimatorDPBM(){
    int b1 = int(r1 + floor(double(m1/batchSize)));
```

```
    int b2 = int(r2 + floor(double(m2/batchSize)));
    int b3 = int(r3 + floor(double(m3/batchSize)));
    int b4 = int(r4 + floor(double(m4/batchSize)));
    ComputeMean(b1);
    estimatorDPBM = 0;
    double s = double(batchSize/4);
    int b = int(floor(double((currentSampleSizeN-batchSize+s)/s)));
    double db = double(b*(double((currentSampleSizeN/batchSize)-1)));
    real sumA = 0, sumB = 0, sumC = 0, sumD = 0, calculation = 0;
    for(int i = 0; i < b1; i++){
        calculation = pow(double((observationVectorA[i]/double(batchSize))-
            mean),2);
        sumA += calculation;
        calculation = 0;
    }
    for(int i = 0; i < b2; i++){
        calculation = pow(double((observationVectorB[i]/double(batchSize))-
            mean),2);
        sumB += calculation;
        calculation = 0;
    }
    for(int i = 0; i < b3; i++){
        calculation = pow(double((observationVectorC[i]/double(batchSize))-
            mean),2);
        sumC += calculation;
        calculation = 0;
    }
    for(int i = 0; i < b4; i++){
        calculation = pow(double((observationVectorD[i]/double(batchSize))-
            mean),2);
        sumD += calculation;
        calculation = 0;
    }
    estimatorDPBM = double((1/db)*double(sumA + sumB + sumC + sumD));
    sumA = 0, sumB = 0, sumC = 0, sumD = 0;
    return;
}

/* Debug tool */
void DynamicBMVarianceEstimator::PrintValuesInVectors(){
    for(int i = 0; i < observationVectorA.size(); i++){
        fprintf(stderr,"value␣in␣A[%d]:␣%f␣\n", i,observationVectorA[i]);
    }
    for(int i = 0; i < observationVectorB.size(); i++){
        fprintf(stderr,"value␣in␣B[%d]:␣%f␣\n", i,observationVectorB[i]);
    }
    for(int i = 0; i < observationVectorC.size(); i++){
        fprintf(stderr,"value␣in␣C[%d]:␣%f␣\n", i,observationVectorC[i]);
    }
    for(int i = 0; i < observationVectorD.size(); i++){
        fprintf(stderr,"value␣in␣D[%d]:␣%f␣\n", i,observationVectorD[i]);
    }
    return;
}
```

# Appendix B

# MSE-DPBM

```
/* Variance estimator based on DPBM by Song */

#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <vector>
#include <cmath>

//include the header file
#include "dpbm_variance_estimator.H"
#include "environment.H"
#include "checkpoint.H"
#include "akaroa/ak_message.H"
#include "akaroa.H"

int checkpointPointer = 0;

DefineVarianceEstimatorType("DPBM", DynamicBMVarianceEstimator)

DynamicBMVarianceEstimator::DynamicBMVarianceEstimator(Environment *env,
    long trans){
    InitializeMethodVariables();
    InitializeMethodVectors();
}

DynamicBMVarianceEstimator::~DynamicBMVarianceEstimator(){
  //destructor
}

void DynamicBMVarianceEstimator::
ProcessObservation(real value)
{
    StartProcessingObservation(value);
}

boolean DynamicBMVarianceEstimator::ReachedCheckpoint(){
    if((checkpointPointer == 1) && (m1 == batchSize)){
```

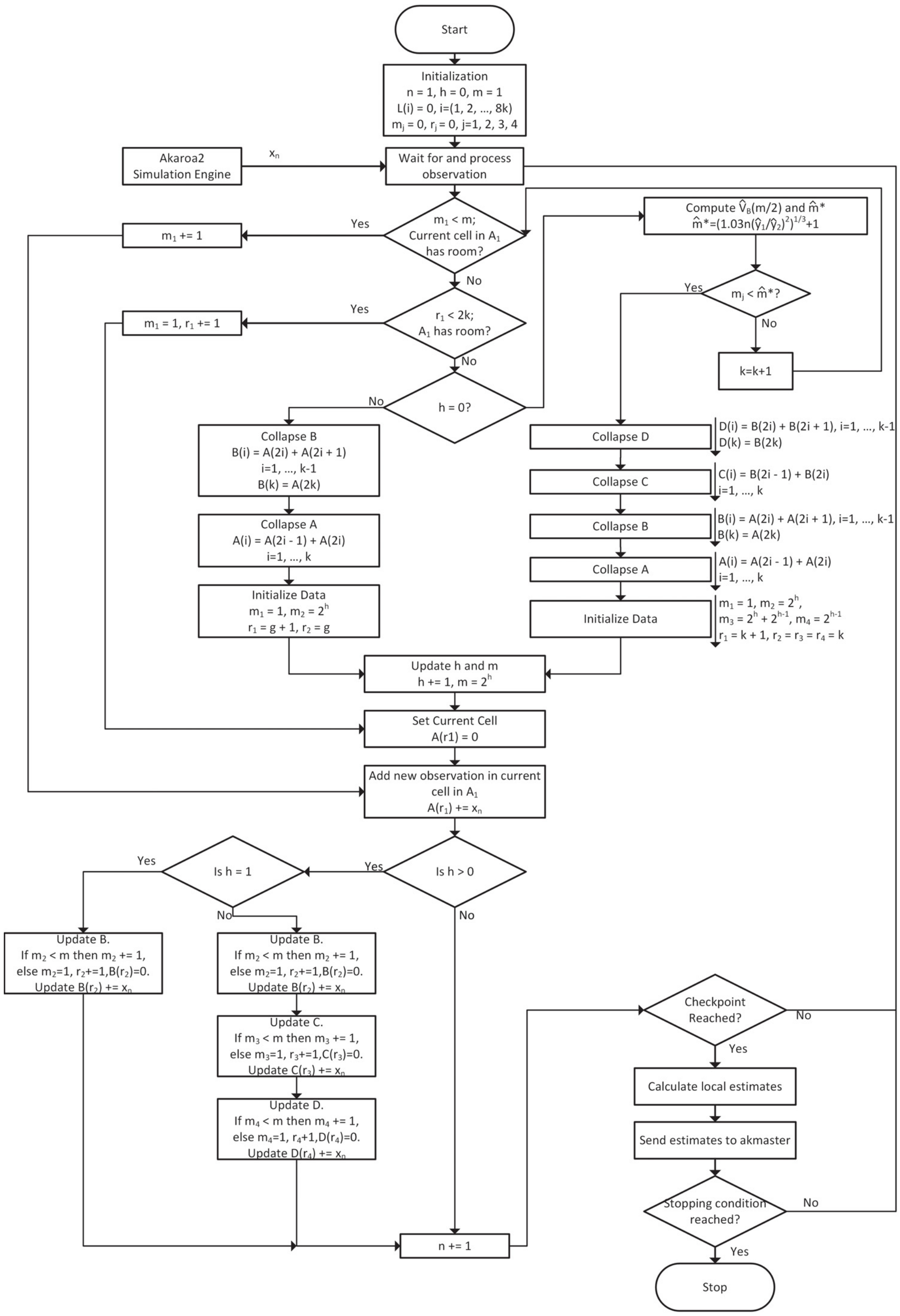


Figure B.1: MSE-DPBM Algorithm as implemented in Akaroa2

```
        ComputeEstimatorDPBM();
        return true;
```

```
    } else {
        return false;
    }
    return false;
}

boolean DynamicBMVarianceEstimator::GetCheckpoint(Checkpoint &cp){
    cp.df = 0;
    cp.mean = mean;
    cp.variance = estimatorDPBM;
    return true;
}

void DynamicBMVarianceEstimator::InitializeMethodVariables(){
    currentSampleSizeN = 1;
    numberOfCollapsesOccured = 0;
    batchSize = pow(2, numberOfCollapsesOccured);
    m1 = 0, m2 = 0, m3 = 0, m4 = 0;
    r1 = 0, r2 = 0, r3 = 0, r4 = 0;
    memorySize = 15;
    estimatorDPBM = 0;
    previousDPBM = 0;
    optimalBatchSize = 0;
    previousMemorySize = 0;
    return;
}

void DynamicBMVarianceEstimator::InitializeMethodVectors(){
    for(int i = previousMemorySize; i < (2*memorySize); i++){
        observationVectorA.push_back(0.00);
        observationVectorB.push_back(0.00);
        observationVectorC.push_back(0.00);
        observationVectorD.push_back(0.00);
    }
    return;
}

void DynamicBMVarianceEstimator::StartProcessingObservation(real value){
    if(m1 < batchSize){
        m1 += 1;
        AddObservation(value);
        EndObservation(value);
        return;
    } else {
        VectorHasRoom(value);
        return;
    }
}

void DynamicBMVarianceEstimator::VectorHasRoom(real value){
    if(r1 < ((2*memorySize)-1)){
        IncreasePointer();
        SetCurrentCell();
        AddObservation(value);
```

```
        EndObservation(value);
        return;
    } else {
        if(numberOfCollapsesOccured == 0){
            CollapseVectorB();
            CollapseVectorA();
            InitializeData();
            UpdateKandM();
            SetCurrentCell();
            AddObservation(value);
            EndObservation(value);
            return;
        } else {
            ComputeOptimalBatchSize();
            checkpointPointer = 1;
            if(batchSize < optimalBatchSize){
                CollapseVectorD();
                CollapseVectorC();
                CollapseVectorB();
                CollapseVectorA();
                InitializeData();
                UpdateKandM();
                SetCurrentCell();
                AddObservation(value);
                EndObservation(value);
                return;
            } else {
                previousMemorySize = 2*memorySize;
                memorySize += 1;
                InitializeMethodVectors();
                VectorHasRoom(value);
            }
        }
    }
}

void DynamicBMVarianceEstimator::InitializeData(){
    if(numberOfCollapsesOccured == 0){
        m1 = 1;
        m2 = pow(2, numberOfCollapsesOccured);
        r1 = memorySize;
        r2 = memorySize - 1;
        return;
    } else {
        m1 = 1;
        m2 = pow(2, numberOfCollapsesOccured);
        m3 = pow(2, numberOfCollapsesOccured) + pow(2, (
            numberOfCollapsesOccured - 1));
        m4 = pow(2, (numberOfCollapsesOccured - 1));
        r1 = memorySize;
        r2 = memorySize - 1;
        r3 = memorySize - 1;
        r4 = memorySize - 1;
        return;
```

```
    }
}

void DynamicBMVarianceEstimator::UpdateKandM(){
    numberOfCollapsesOccured += 1;
    batchSize = pow(2,numberOfCollapsesOccured);
    return;
}

void DynamicBMVarianceEstimator::SetCurrentCell(){
    observationVectorA[r1] = 0;
    return;
}

void DynamicBMVarianceEstimator::CollapseVectorD(){
    for(int i = 1; i <= (memorySize-1); i++){
        observationVectorD[i-1] = observationVectorB[(2*i)-1] +
            observationVectorB[2*i];
    }
    observationVectorD[memorySize-1] = observationVectorB[(2*memorySize)-1];
    m4 = m2;
    return;
}

void DynamicBMVarianceEstimator::CollapseVectorC(){
    for(int i = 1; i <= (memorySize);i++){
        observationVectorC[i-1] = (observationVectorB[(2*i)-2] +
            observationVectorB[(2*i)-1]);
    }
    m3 = m2 * pow(2, numberOfCollapsesOccured);
    return;
}

void DynamicBMVarianceEstimator::CollapseVectorB(){
    for(int i = 1; i <= (memorySize-1); i++){
        observationVectorB[i-1] = observationVectorA[(2*i)-1] +
            observationVectorA[2*i];
    }
    observationVectorB[memorySize-1] = observationVectorA[(2*memorySize)-1];
    return;
}

void DynamicBMVarianceEstimator::CollapseVectorA(){
    for(int i = 1; i <= (memorySize);i++){
        observationVectorA[i-1] = (observationVectorA[(2*i)-2] +
            observationVectorA[(2*i)-1]);
    }
    return;
}

void DynamicBMVarianceEstimator::IncreasePointer(){
    m1 = 1;
    r1 += 1;
    return;
```

```
}

void DynamicBMVarianceEstimator::EndObservation(real value){
    if(numberOfCollapsesOccured == 1){
        UpdateB(value);
    }else if(numberOfCollapsesOccured > 1){
        UpdateD(value);
        UpdateC(value);
        UpdateB(value);
    }
    currentSampleSizeN += 1;
    return;
}

void DynamicBMVarianceEstimator::UpdateB(real value){
    if(m2 < batchSize){
        m2 += 1;
    } else {
        m2 = 1;
        r2 += 1;
        observationVectorB[r2] = 0;
    }
    observationVectorB[r2] += value;
    return;
}

void DynamicBMVarianceEstimator::UpdateC(real value){
    if(m3 < batchSize){
        m3 += 1;
    } else {
        m3 = 1;
        r3 += 1;
        observationVectorC[r3] = 0;
    }
    observationVectorC[r3] += value;
    return;
}

void DynamicBMVarianceEstimator::UpdateD(real value){
    if(m4 < batchSize){
        m4 += 1;
    } else {
        m4 = 1;
        r4 += 1;
        observationVectorD[r4] = 0;
    }
    observationVectorD[r4] += value;
    return;
}

void DynamicBMVarianceEstimator::AddObservation(real value){
    observationVectorA[r1] += value;
    return;
}
```

```
void DynamicBMVarianceEstimator::ComputeOptimalBatchSize(){
    ComputeEstimatorDPBM();
    ComputeVirtualA();
    ComputeVirtualB();
    ComputeEstimatorBDPBM();
    real gamma0 = ((double(currentSampleSizeN)*estimatorBDPBM)/
        sampleVariance);
    real gamma1 = (double(((double(currentSampleSizeN)*double(batchSize))*(
        estimatorDPBM-estimatorBDPBM)))/sampleVariance);
    optimalBatchSize = cbrt((1.12*double(currentSampleSizeN))*(pow(double(
        gamma1/gamma0),2)))+1;
    return;
}

void DynamicBMVarianceEstimator::ComputeMean(int b1){
    mean = 0;
    for(int i = 0; i < b1; i++){
        mean += (observationVectorA[i]);
    }
    mean = (mean/double(currentSampleSizeN));
    real sum = 0, calculation = 0;
    //sample variance
    for(int i = 0; i < b1; i++){
        calculation = pow(observationVectorA[i]-mean, 2);
        sum += calculation;
        calculation = 0;
    }
    sampleVariance = (sum / double(currentSampleSizeN));
    return;
}

void DynamicBMVarianceEstimator::ComputeEstimatorDPBM(){
    int b1 = int(r1 + floor(double(m1/batchSize)));
    int b2 = int(r2 + floor(double(m2/batchSize)));
    int b3 = int(r3 + floor(double(m3/batchSize)));
    int b4 = int(r4 + floor(double(m4/batchSize)));
    ComputeMean(b1);
    estimatorDPBM = 0;
    double s = floor(double(batchSize/4));
    int b = int(floor(double((currentSampleSizeN-batchSize+s)/s)));
    double db = double(b*(double((currentSampleSizeN/batchSize)-1)));
    real sumA = 0, sumB = 0, sumC = 0, sumD = 0, calculation = 0;
    for(int i = 0; i < b1; i++){
        calculation = pow(double((observationVectorA[i]/double(batchSize))-
            mean),2);
        sumA += calculation;
        calculation = 0;
    }
    for(int i = 0; i < b2; i++){
        calculation = pow(double((observationVectorB[i]/double(batchSize))-
            mean),2);
        sumB += calculation;
        calculation = 0;
```

```
    }
    for(int i = 0; i < b3; i++){
        calculation = pow(double((observationVectorC[i]/double(batchSize))-
            mean),2);
        sumC += calculation;
        calculation = 0;
    }
    for(int i = 0; i < b4; i++){
        calculation = pow(double((observationVectorD[i]/double(batchSize))-
            mean),2);
        sumD += calculation;
        calculation = 0;
    }
    estimatorDPBM = double((1/db)*double(sumA + sumB + sumC + sumD));
    sumA = 0, sumB = 0, sumC = 0, sumD = 0;
    return;
}

void DynamicBMVarianceEstimator::ComputeVirtualA(){
    vectorBDPBM1.reserve(4*memorySize);
    real sumA = 0, sumB = 0;
    for(int i = 1; i <= ceil(double(currentSampleSizeN)/double(batchSize));
        i++){
        for(int k = (i-1); k < r1; k++){
            sumA += observationVectorA[k];
        }
        for(int k = (i-1); k <= r2; k++){
            sumB += observationVectorB[k];
        }
        vectorBDPBM1[(2*i)-2] = sumA-sumB;
        sumA = 0;
        sumB = 0;
    }
    for(int i = 1; i <= floor((double(currentSampleSizeN)/double(batchSize))
        +0.5);i++){
        vectorBDPBM1[(2*i)-1] = observationVectorA[i-1] - vectorBDPBM1[(2*i)
            -2];
    }
    return;
}

void DynamicBMVarianceEstimator::ComputeVirtualB(){
    vectorBDPBM2.reserve(4*memorySize);
    real sumA = 0, sumB = 0;
    for(int i = 1; i <= ceil((double(currentSampleSizeN)-(double(batchSize)
        /4))/double(batchSize)) ; i++){
        for(int k = (i-1); k <= r3; k++){
            sumA += observationVectorC[k];
        }
        for(int k = (i-1); k <= r4; k++){
            sumB += observationVectorD[k];
        }
        vectorBDPBM2[(2*i)-2] = sumA-sumB;
        sumA = 0;
```

```
        sumB = 0;
    }
    for(int i = 1; i <= floor(((double(currentSampleSizeN)-(double(batchSize)/4))/double(batchSize))+0.5);i++){
        vectorBDPBM2[(2*i)-1] = observationVectorC[i-1] - vectorBDPBM2[(2*i)-2];
    }
    return;
}

void DynamicBMVarianceEstimator::ComputeEstimatorBDPBM(){
    estimatorBDPBM = 0;
    int previousBatchSize = batchSize / 2;
    double s = (double(previousBatchSize)/2);
    int b = floor((double(currentSampleSizeN)-double(previousBatchSize)+s)/s);
    double db = b*((double(currentSampleSizeN)/double(previousBatchSize))-1);
    double sumA = 0, sumB = 0, calculation = 0;
    int bA = floor(double(currentSampleSizeN)/double(previousBatchSize));
    int bB = floor((double(currentSampleSizeN)-(double(previousBatchSize)/2))/double(previousBatchSize));
    //sleep(1);
    for(int i = 1; i <= bA; i++){
        calculation = pow((((vectorBDPBM1[i-1])/double(previousBatchSize))-mean),2);
        sumA += calculation;
        calculation = 0;
    }
    for(int i = 1; i <= bB; i++){
        calculation = pow((((vectorBDPBM2[i-1])/double(previousBatchSize))-mean),2);
        sumB += calculation;
        calculation = 0;
    }
    estimatorBDPBM = (1/db)*(sumA + sumB);
    sumA = 0;
    sumB = 0;
    return;
}

void DynamicBMVarianceEstimator::PrintValuesInVectors(){
    for(int i = 0; i < observationVectorA.size(); i++){
        fprintf(stderr,"value␣in␣A[%d]:␣%f␣\n", i,observationVectorA[i]);
    }
    for(int i = 0; i < observationVectorB.size(); i++){
        fprintf(stderr,"value␣in␣B[%d]:␣%f␣\n", i,observationVectorB[i]);
    }
    for(int i = 0; i < observationVectorC.size(); i++){
        fprintf(stderr,"value␣in␣C[%d]:␣%f␣\n", i,observationVectorC[i]);
    }
    for(int i = 0; i < observationVectorD.size(); i++){
        fprintf(stderr,"value␣in␣D[%d]:␣%f␣\n", i,observationVectorD[i]);
    }
```

```
    return;
}
```

# Appendix C

# Modified MSE-DPBM

```
/* Modified MSE-DPBM */

#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <vector>
#include <cmath>

//include the header file
#include "dpbm_variance_estimator.H"
#include "environment.H"
#include "checkpoint.H"
#include "akaroa/ak_message.H"
#include "akaroa.H"

int checkpointPointer = 0, newBatch = 0;

DefineVarianceEstimatorType("DPBM", DynamicBMVarianceEstimator)

DynamicBMVarianceEstimator::DynamicBMVarianceEstimator(Environment *env,
    long trans){
    InitializeMethodVariables();
    InitializeMethodVectors();
}

DynamicBMVarianceEstimator::~DynamicBMVarianceEstimator(){
  //destructor
}

void DynamicBMVarianceEstimator::
ProcessObservation(real value)
{
    StartProcessingObservation(value);
}

boolean DynamicBMVarianceEstimator::ReachedCheckpoint(){
    if((checkpointPointer == 1) && (m1 == batchSize)){
```

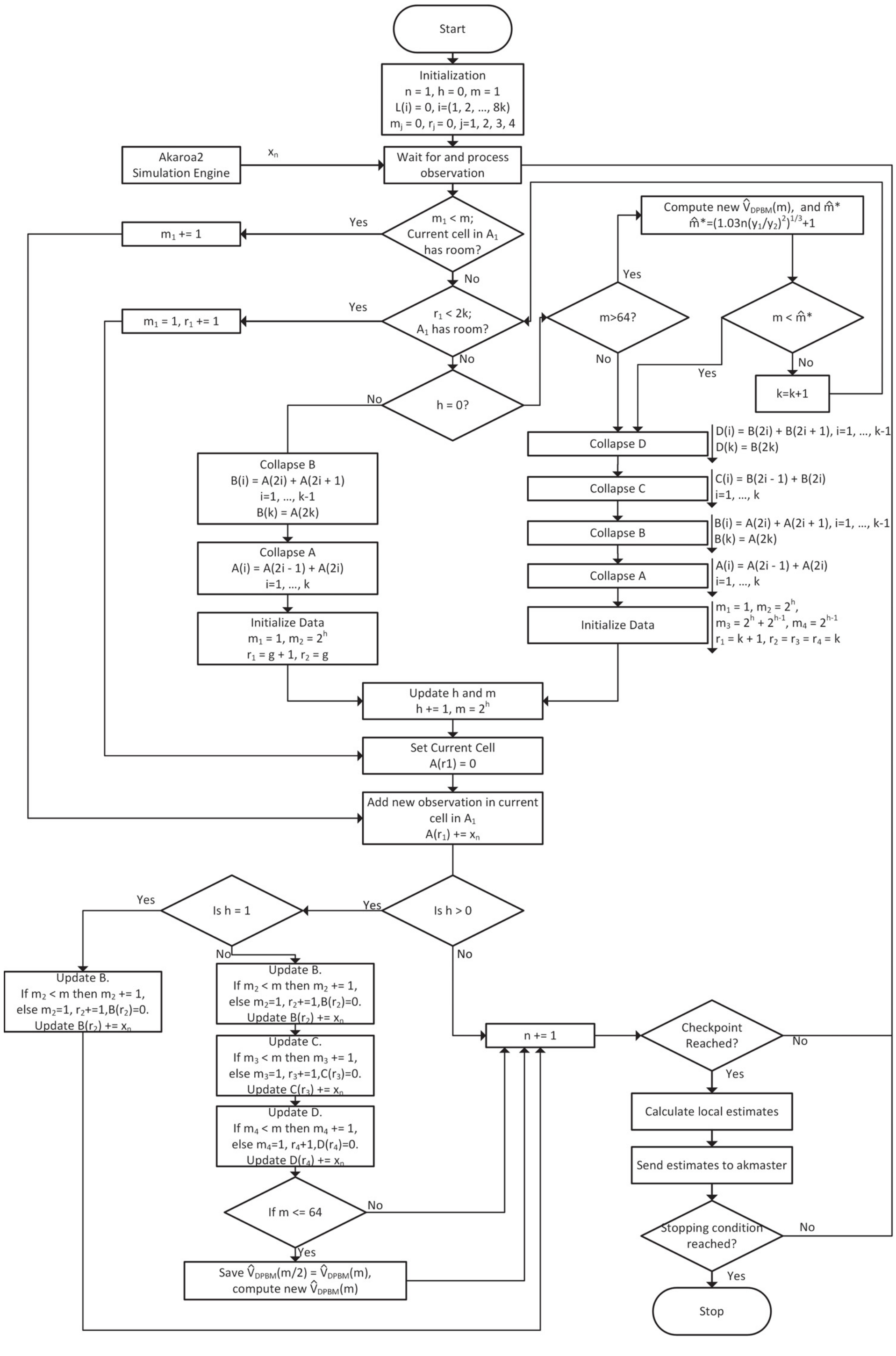


Figure C.1: Modified MSE-DPBM Algorithm as implemented in Akaroa2

```
        pointerToDPBM = 1;
        ComputeEstimatorDPBM();
        return true;
    } else {
        return false;
    }
    return false;
}

boolean DynamicBMVarianceEstimator::GetCheckpoint(Checkpoint &cp){
    cp.df = 0;
    cp.mean = mean;
    cp.variance = submitDPBM;
    return true;
}

void DynamicBMVarianceEstimator::InitializeMethodVariables(){
    currentSampleSizeN = 1;
    numberOfCollapsesOccured = 0;
    batchSize = pow(2, numberOfCollapsesOccured);
    m1 = 0, m2 = 0, m3 = 0, m4 = 0;
    r1 = 0, r2 = 0, r3 = 0, r4 = 0;
    memorySize = 15;
    estimatorDPBM = 0;
    previousDPBM = 0;
    optimalBatchSize = 0;
    previousMemorySize = 0;
    return;
}

void DynamicBMVarianceEstimator::InitializeMethodVectors(){
    for(int i = previousMemorySize; i < (2*memorySize); i++){
        observationVectorA.push_back(0.00);
        observationVectorB.push_back(0.00);
        observationVectorC.push_back(0.00);
        observationVectorD.push_back(0.00);
    }
    return;
}

void DynamicBMVarianceEstimator::StartProcessingObservation(real value){
    if(m1 < batchSize){
        m1 += 1;
        AddObservation(value);
        EndObservation(value);
        return;
    } else {
        VectorHasRoom(value);
        return;
    }
}

void DynamicBMVarianceEstimator::VectorHasRoom(real value){
    if(r1 < ((2*memorySize)-1)){
```

```
        IncreasePointer();
        SetCurrentCell();
        AddObservation(value);
        EndObservation(value);
        return;
    } else {
        if(numberOfCollapsesOccured == 0){
            CollapseVectorB();
            CollapseVectorA();
            InitializeData();
            UpdateKandM();
            SetCurrentCell();
            AddObservation(value);
            EndObservation(value);
            return;
        } else {
            if(batchSize < 65) {
                CollapseVectorD();
                CollapseVectorC();
                CollapseVectorB();
                CollapseVectorA();
                InitializeData();
                UpdateKandM();
                SetCurrentCell();
                AddObservation(value);
                EndObservation(value);
                previousDPBM = estimatorDPBM;
                ComputeEstimatorDPBM();
                return;
            } else {
                ComputeOptimalBatchSize();
                checkpointPointer = 1;
                if(batchSize < optimalBatchSize){
                    CollapseVectorD();
                    CollapseVectorC();
                    CollapseVectorB();
                    CollapseVectorA();
                    InitializeData();
                    UpdateKandM();
                    SetCurrentCell();
                    AddObservation(value);
                    EndObservation(value);
                    return;
                } else {
                    previousMemorySize = 2*memorySize;
                    memorySize += 1;
                    InitializeMethodVectors();
                    VectorHasRoom(value);
                }
            }

        }
    }
}
```

```

void DynamicBMVarianceEstimator::InitializeData(){
    if(numberOfCollapsesOccured == 0){
        m1 = 1;
        m2 = pow(2, numberOfCollapsesOccured);
        r1 = memorySize;
        r2 = memorySize - 1;
        return;
    } else {
        m1 = 1;
        m2 = pow(2, numberOfCollapsesOccured);
        m3 = pow(2, numberOfCollapsesOccured) + pow(2, (
            numberOfCollapsesOccured - 1));
        m4 = pow(2, (numberOfCollapsesOccured - 1));
        r1 = memorySize;
        r2 = memorySize - 1;
        r3 = memorySize - 1;
        r4 = memorySize - 1;
        return;
    }
}

void DynamicBMVarianceEstimator::UpdateKandM(){
    numberOfCollapsesOccured += 1;
    batchSize = pow(2,numberOfCollapsesOccured);
    return;
}

void DynamicBMVarianceEstimator::SetCurrentCell(){
    observationVectorA[r1] = 0;
    return;
}

void DynamicBMVarianceEstimator::CollapseVectorD(){
    for(int i = 1; i <= (memorySize-1); i++){
        observationVectorD[i-1] = observationVectorB[(2*i)-1] +
            observationVectorB[2*i];
    }
    observationVectorD[memorySize-1] = observationVectorB[(2*memorySize)-1];
    m4 = m2;
    return;
}

void DynamicBMVarianceEstimator::CollapseVectorC(){
    for(int i = 1; i <= (memorySize);i++){
        observationVectorC[i-1] = (observationVectorB[(2*i)-2] +
            observationVectorB[(2*i)-1]);
    }
    m3 = m2 * pow(2, numberOfCollapsesOccured);
    return;
}

void DynamicBMVarianceEstimator::CollapseVectorB(){
    for(int i = 1; i <= (memorySize-1); i++){
```

```
        observationVectorB[i-1] = observationVectorA[(2*i)-1] +
            observationVectorA[2*i];
    }
    observationVectorB[memorySize-1] = observationVectorA[(2*memorySize)-1];
    return;
}

void DynamicBMVarianceEstimator::CollapseVectorA(){
    for(int i = 1; i <= (memorySize);i++){
        observationVectorA[i-1] = (observationVectorA[(2*i)-2] +
            observationVectorA[(2*i)-1]);
    }
    return;
}

void DynamicBMVarianceEstimator::IncreasePointer(){
    m1 = 1;
    r1 += 1;
    return;
}

void DynamicBMVarianceEstimator::EndObservation(real value){
    if(numberOfCollapsesOccured == 1){
        UpdateB(value);
    }else if(numberOfCollapsesOccured > 1){
        UpdateD(value);
        UpdateC(value);
        UpdateB(value);
    }
    currentSampleSizeN += 1;
    return;
}

void DynamicBMVarianceEstimator::UpdateB(real value){
    if(m2 < batchSize){
        m2 += 1;
    } else {
        m2 = 1;
        r2 += 1;
        observationVectorB[r2] = 0;
    }
    observationVectorB[r2] += value;
    return;
}

void DynamicBMVarianceEstimator::UpdateC(real value){
    if(m3 < batchSize){
        m3 += 1;
    } else {
        m3 = 1;
        r3 += 1;
        observationVectorC[r3] = 0;
    }
    observationVectorC[r3] += value;
```

```
    return;
}

void DynamicBMVarianceEstimator::UpdateD(real value){
    if(m4 < batchSize){
        m4 += 1;
    } else {
        m4 = 1;
        r4 += 1;
        observationVectorD[r4] = 0;
    }
    observationVectorD[r4] += value;
    return;
}

void DynamicBMVarianceEstimator::AddObservation(real value){
    observationVectorA[r1] += value;
    return;
}

void DynamicBMVarianceEstimator::ComputeOptimalBatchSize(){
    previousDPBM = estimatorDPBM;
    ComputeEstimatorDPBM();
    real gamma0 = ((double(currentSampleSizeN)*estimatorDPBM)/sampleVariance
        );
    real gamma1 = (double(((double(currentSampleSizeN)*double(batchSize))*(
        estimatorDPBM-previousDPBM)))/sampleVariance);
    optimalBatchSize = cbrt((1.03*double(currentSampleSizeN))*(pow(double(
        gamma1/gamma0),2)))+1;
    return;
}

void DynamicBMVarianceEstimator::ComputeMean(int b1){
    mean = 0;
    for(int i = 0; i < b1; i++){
        mean += (observationVectorA[i]);
    }
    mean = (mean/double(currentSampleSizeN));
    real sum = 0, calculation = 0;
    for(int i = 0; i < b1; i++){
        calculation = pow(observationVectorA[i]-mean, 2);
        sum += calculation;
        calculation = 0;
    }
    sampleVariance = (sum / double(currentSampleSizeN));
    return;
}

void DynamicBMVarianceEstimator::ComputeEstimatorDPBM(){
    if(pointerToDPBM == 1){
        int b1 = int(r1 + floor(double(m1/batchSize)));
        int b2 = int(r2 + floor(double(m2/batchSize)));
        int b3 = int(r3 + floor(double(m3/batchSize)));
        int b4 = int(r4 + floor(double(m4/batchSize)));
```

```
        ComputeMean(b1);
        double s = double(batchSize/4);
        int b = int(floor(double((currentSampleSizeN-batchSize+s)/s)));
        double db = double(b*(double((currentSampleSizeN/batchSize)-1)));
        real sumA = 0, sumB = 0, sumC = 0, sumD = 0, calculation = 0;
        for(int i = 0; i < b1; i++){
            calculation = pow(double((observationVectorA[i]/double(batchSize))-mean),2);
            sumA += calculation;
            calculation = 0;
        }
        for(int i = 0; i < b2; i++){
            calculation = pow(double((observationVectorB[i]/double(batchSize))-mean),2);
            sumB += calculation;
            calculation = 0;
        }
        for(int i = 0; i < b3; i++){
            calculation = pow(double((observationVectorC[i]/double(batchSize))-mean),2);
            sumC += calculation;
            calculation = 0;
        }
        for(int i = 0; i < b4; i++){
            calculation = pow(double((observationVectorD[i]/double(batchSize))-mean),2);
            sumD += calculation;
            calculation = 0;
        }
        submitDPBM = double((1/db)*double(sumA + sumB + sumC + sumD));
        sumA = 0, sumB = 0, sumC = 0, sumD = 0;
        pointerToDPBM = 0;
        return;
    } else {
        int b1 = int(r1 + floor(double(m1/batchSize)));
        int b2 = int(r2 + floor(double(m2/batchSize)));
        int b3 = int(r3 + floor(double(m3/batchSize)));
        int b4 = int(r4 + floor(double(m4/batchSize)));
        ComputeMean(b1);
        estimatorDPBM = 0;
        double s = double(batchSize/4);
        int b = int(floor(double((currentSampleSizeN-batchSize+s)/s)));
        double db = double(b*(double((currentSampleSizeN/batchSize)-1)));
        real sumA = 0, sumB = 0, sumC = 0, sumD = 0, calculation = 0;
        for(int i = 0; i < b1; i++){
            calculation = pow(double((observationVectorA[i]/double(batchSize))-mean),2);
            sumA += calculation;
            calculation = 0;
        }
        for(int i = 0; i < b2; i++){
            calculation = pow(double((observationVectorB[i]/double(batchSize))-mean),2);
            sumB += calculation;
```

```
            calculation = 0;
        }
        for(int i = 0; i < b3; i++){
            calculation = pow(double((observationVectorC[i]/double(batchSize
                ))-mean),2);
            sumC += calculation;
            calculation = 0;
        }
        for(int i = 0; i < b4; i++){
            calculation = pow(double((observationVectorD[i]/double(batchSize
                ))-mean),2);
            sumD += calculation;
            calculation = 0;
        }
        estimatorDPBM = double((1/db)*double(sumA + sumB + sumC + sumD));
        sumA = 0, sumB = 0, sumC = 0, sumD = 0;
        return;
    }
}

void DynamicBMVarianceEstimator::PrintValuesInVectors(){
    for(int i = 0; i < observationVectorA.size(); i++){
        fprintf(stderr,"value␣in␣A[%d]:␣%f␣\n", i,observationVectorA[i]);
    }
    for(int i = 0; i < observationVectorB.size(); i++){
        fprintf(stderr,"value␣in␣B[%d]:␣%f␣\n", i,observationVectorB[i]);
    }
    for(int i = 0; i < observationVectorC.size(); i++){
        fprintf(stderr,"value␣in␣C[%d]:␣%f␣\n", i,observationVectorC[i]);
    }
    for(int i = 0; i < observationVectorD.size(); i++){
        fprintf(stderr,"value␣in␣D[%d]:␣%f␣\n", i,observationVectorD[i]);
    }
    return;
}
```

# Appendix D

# Coverage Analysis

```
#!/usr/bin/perl
use warnings;
use strict;
use DBI;
use POSIX qw(strftime);
$| = 0;

our ($model, $load, $method, $transMethod, $seed, $theoreticalMean,
     $currentTable, $mean, $standardDeviation, $lMin, $deltaCoverage);
our ($HOST,$MYSQLPORT,$DATABASE,$USERNAME,$PASSWD, $dbh);
our ($badCI, $totalCI);
our ($badLength, $goodLength) = 0;
our ($goodLBadCi, $goodLGoodCi, $goodLenghtTotal, $coverage);

require "db.conf";

sub InitializeVariables {
        $model = $ARGV[0];
        $load = $ARGV[1];
        $method = $ARGV[2];
        $transMethod = $ARGV[3];
        $seed = "128:0";
        GetTheoreticalMean();
        return;
}

sub GetTheoreticalMean{
        $theoreticalMean = `$model $load -t`;
        return;
}

sub CreateDatabase{
        my $date = strftime "%m%d%Y", localtime;
        my @dbLoad = split(/\./,$load);
        $currentTable = "O_".$method."_".$model."_".$dbLoad[0].$dbLoad[1]."_
            ".$date."_".$transMethod;
        #print $currentTable."\n";
```

```
        my $createTableQuery = $dbh->prepare("CREATE␣TABLE␣IF␣NOT␣EXISTS␣$currentTable␣(
␣␣␣␣␣␣␣␣`id`␣bigint␣primary␣key␣auto_increment,
␣␣`estimate`␣real␣signed␣NOT␣NULL,
␣␣`delta`␣float␣unsigned␣NOT␣NULL,
␣␣`confidence`␣float␣unsigned␣NOT␣NULL,
␣␣`totalObs`␣bigint␣unsigned␣NOT␣NULL,
␣␣`transObs`␣int␣unsigned␣NOT␣NULL,
␣␣`seed`␣text␣NOT␣NULL,
␣␣`BoolCI`␣int␣NOT␣NULL,
␣␣`BoolLength`␣int␣NOT␣NULL
)␣ENGINE=InnoDB␣DEFAULT␣CHARSET=utf8␣COLLATE=utf8_czech_ci;");
        $createTableQuery->execute();
        $createTableQuery->finish();
        return;
}

sub ConnectToDatabase{
$dbh = DBI->connect("dbi:mysql:$DATABASE:$HOST:$MYSQLPORT",
                        $USERNAME,
                        $PASSWD,
                        { 'PrintError' => 1, 'RaiseError' => 1 }
                        ) or die $DBI::errstr;
                        return;
}

sub SaveIntoDatabase{
        my ($estimate, $delta, $conf, $totalObs, $transObs, $pointerCI, $pointerLenght) = @_;
        my $insertQuery = $dbh->prepare("INSERT␣INTO␣$currentTable␣(estimate,delta,confidence,totalObs,
␣␣␣␣␣␣␣␣transObs,␣seed,␣BoolCI,␣BoolLength)␣values␣(?,␣?,␣?,␣?,␣?,␣?,␣?,␣?)");
        $insertQuery->execute($estimate, $delta, $conf, $totalObs, $transObs, $seed, $pointerCI, $pointerLenght);
        $insertQuery->finish();
        return;
}

sub FirstRun{
        $badCI = 0;
        while($badCI < 200){
                my $output = `akrun -n 1 -s -r $seed -D AnalysisMethod=$method -D TransientMethod=$transMethod $model $load`;
                #print $output."\n";
                my $regEx = "([+-]?\\d*\\.\\d+)(?![-+0-9\\.]).*?([+-]?\\d*\\.\\d+)(?![-+0-9\\.]).*?([+-]?\\d*\\.\\d+)(?![-+0-9\\.]).*?(\\d+).*?(\\d+)";
                $output =~ m/$regEx/is;
                my $estimate = $1;
                my $delta = $2;
                my $conf = $3;
                my $totalObs = $4;
                my $transObs = $5;
```

```
                if(!defined($delta)){
                        my $regEx = "(\\d+)(\\s+)(\\d+).*?([+-]?\\d*\\.\\d+)
                            (?![-+0-9\\.]).*?([+-]?\\d*\\.\\d+)
                            (?![-+0-9\\.]).*?(\\d+).*?(\\d+)";
                        $output =~ m/$regEx/is;
                        $estimate = $3;
                        $delta = $4;
                        $conf = $5;
                        $totalObs = $6;
                        $transObs = $7;
                }
                if(($theoreticalMean >= ($estimate - $delta)) && (
                    $theoreticalMean <= ($estimate+$delta))){ #is covered
                    by the CI
                        SaveIntoDatabase($estimate, $delta, $conf, $totalObs
                            , $transObs, 1, 0);
                } else { #is not covered by the CI
                        $badCI +=1;
                        SaveIntoDatabase($estimate, $delta, $conf, $totalObs
                            , $transObs, 0, 0);
                }
                $output =~ m/RandomNumberState: (\d.*)/;
                $seed = $1;
                $totalCI += 1; #keeps the total number of runs, will need to
                     be recalculated after some runs are rejected (not long
                     enuf)
        }
        CalculateStandardDeviation();
        RejectShortRuns();
        return;
}

sub RejectShortRuns{
        my $selectQuery = $dbh->prepare("SELECT␣*␣FROM␣$currentTable");
        $selectQuery->execute();
        my @row;
        while (@row = $selectQuery->fetchrow_array())
        {
                if($row[4] < $lMin){
                        my $updateQuery = $dbh->prepare("UPDATE␣
                            $currentTable␣SET␣BoolLength␣=␣?␣WHERE␣(id␣=␣?)
                            ");
                        $updateQuery->execute('0',$row[0]);
                        $updateQuery->finish();
                        #print "update␣id␣=␣$row[0]␣with␣lenght:␣$row[4]␣<␣
                            $lMin␣\n";
                        $badLength += 1;
                } else {
                        my $updateQuery = $dbh->prepare("UPDATE␣
                            $currentTable␣SET␣BoolLength␣=␣?␣WHERE␣(id␣=␣?)
                            ");
                        $updateQuery->execute('1',$row[0]);
                        $updateQuery->finish();
                        #print "update␣id␣=␣$row[0]␣with␣lenght:␣$row[4]␣>␣
```

```
                            $1Min␣\n";
                        $goodLength += 1;
                }
        }
        $selectQuery->finish();
        #print "now␣we␣have␣good:␣$countGood␣and␣bad:␣$countBad␣␣and␣total:␣
            $totalCI␣\n";
        return;
}

sub CalculateStandardDeviation{
        my $selectQuery = $dbh->prepare("SELECT␣*␣FROM␣$currentTable");
        $selectQuery->execute();
        my @row;
        $mean = 0;
        my $sum = 0;
        my $count = 0;
        while (@row = $selectQuery->fetchrow_array())
        {
                $mean += $row[4];
                $count++;
        }
        $selectQuery -> finish();
        $mean = $mean/$count;
        #print "Mean␣is:␣$mean␣\n";
        $selectQuery->execute();
        while (@row = $selectQuery->fetchrow_array())
        {
                $sum += (($row[4] - $mean)**2);
                #print "$sum␣=␣$row[4]␣-␣$mean␣squared␣\n";
        }
        $standardDeviation = sqrt((1/($totalCI-1))*($sum));
        $1Min = $mean - $standardDeviation;
        my $insertQuery = $dbh->prepare("INSERT␣INTO␣`".$model."_results`␣(`
            model`,`load`,`analysisMethod`,`transientMethod`,`
            standardDeviation`,`mean`)␣values␣(?,␣?,␣?,␣?,␣?,␣?)");
        $insertQuery->execute($model, $load, "O_".$method, $transMethod,
            $standardDeviation, $mean);
        $insertQuery->finish();
        #print "ST:␣$standardDeviation␣and␣mean:␣$mean␣and␣equals␣to:␣".
            $1Min."\n";
        $selectQuery -> finish();
        return;
}

sub FirstCheckpoint{
        #get 200 of good lenght, but not covering the mean and calculate
            coverage
        my $goodLengthBadCi = 0;
        my $selectQuery = $dbh->prepare("SELECT␣*␣FROM␣$currentTable");
        $selectQuery->execute();
        my @row;
        my $count;
        while (@row = $selectQuery->fetchrow_array())
```

```
	{
		if(($row[7] == 0) && ($row[8] == 1)){
			$goodLengthBadCi +=1 ;
		}
	}
	my $numOfRuns = 0;
	while($goodLengthBadCi < 200){
		my $output = 0;
		if($numOfRuns < 1000000){
			$output = `akrun -n 1 -s -r $seed -D AnalysisMethod=$method -D TransientMethod=$transMethod $model $load`;
			$numOfRuns += 1;
		} else {
			print "Run length achieved, did not record 200 bad CI with good length, exiting. \n";
			exit;
		}
		#print $output."\n";
		#parse the output of simulation
		my $regEx = "([+-]?\\d*\\.\\d+)(?![-+0-9\\.]).*?([+-]?\\d*\\.\\d+)(?![-+0-9\\.]).*?([+-]?\\d*\\.\\d+)(?![-+0-9\\.]).*?(\\d+).*?(\\d+)";
		$output =~ m/$regEx/is;
		my $estimate = $1;
		my $delta = $2;
		my $conf = $3;
		my $totalObs = $4;
		my $transObs = $5;
		if(!defined($delta)){
			my $regEx = "(\\d+)(\\s+)(\\d+).*?([+-]?\\d*\\.\\d+)(?![-+0-9\\.]).*?([+-]?\\d*\\.\\d+)(?![-+0-9\\.]).*?(\\d+).*?(\\d+)";
			$output =~ m/$regEx/is;
			$estimate = $3;
			$delta = $4;
			$conf = $5;
			$totalObs = $6;
			$transObs = $7;
		}
		if($totalObs > $lMin){
			if(($theoreticalMean >= ($estimate - $delta)) && ($theoreticalMean <= ($estimate+$delta))){ #is covered by the CI
				SaveIntoDatabase($estimate, $delta, $conf, $totalObs, $transObs, 1, 1);
			} else { #is not covered by the CI
				$goodLengthBadCi += 1;
				SaveIntoDatabase($estimate, $delta, $conf, $totalObs, $transObs, 0, 1);
			}
		} else {
			$badLength += 1;
		}
```

```
                $output =~ m/RandomNumberState: (\d.*)/;
                $seed = $1;
        }
        CalculateDeltaOfCoverage();
        return;
}

sub CalculateDeltaOfCoverage {
        my $selectQuery = $dbh->prepare("SELECT * FROM $currentTable");
        $selectQuery->execute();
        my @row;
        my $z = 1.96;
        ($goodLBadCi, $goodLGoodCi, $goodLenghtTotal) = 0;
        while (@row = $selectQuery->fetchrow_array())
        {
                if(($row[7] == 0) && ($row[8] == 1)){
                        $goodLBadCi +=1;
                        $goodLenghtTotal +=1;
                } elsif(($row[7] == 1) && ($row[8] == 1)){
                        $goodLGoodCi += 1;
                        $goodLenghtTotal +=1;
                }
        }
        $selectQuery->finish();
        $coverage = $goodLGoodCi / $goodLenghtTotal;
        if($goodLenghtTotal > 99){
                $z = 1.96;
        } else {
                print "need to use student t distro \n";
                $z = 1.96;
        }
        $deltaCoverage = $z * sqrt(($coverage * (1- $coverage))/(
            $goodLenghtTotal));
        #print "good = $goodLGoodCi, total = $goodLenghtTotal, coverage = 
            $coverage, delta_coverage = $deltaCoverage \n";
        #print "delta of coverage is: ".$deltaCoverage."\n";
        return;
}

sub SequentialAnalysis{
        while($deltaCoverage > 0.01) {
                my $output = `akrun -n 1 -s -r $seed -D AnalysisMethod=
                    $method -D TransientMethod=$transMethod $model $load`;
                #parse the output of simulation
                my $regEx = "([+-]?\\d*\\.\\d+)(?![-+0-9\\.]).*?([+-]?\\d
                    *\\.\\d+)(?![-+0-9\\.]).*?([+-]?\\d*\\.\\d+)
                    (?![-+0-9\\.]).*?(\\d+).*?(\\d+)";
                $output =~ m/$regEx/is;
                my $estimate = $1;
                my $delta = $2;
                my $conf = $3;
                my $totalObs = $4;
                my $transObs = $5;
                if(!defined($delta)){
```

```
                        my $regEx = "(\\d+)(\\s+)(\\d+).*?([+-]?\\d*\\.\\d+)
                            (?![-+0-9\\.]).*?([+-]?\\d*\\.\\d+)
                            (?![-+0-9\\.]).*?(\\d+).*?(\\d+)";
                        $output =~ m/$regEx/is;
                        $estimate = $3;
                        $delta = $4;
                        $conf = $5;
                        $totalObs = $6;
                        $transObs = $7;
                }
                if($totalObs > $lMin){
                        if(($theoreticalMean >= ($estimate - $delta)) && (
                            $theoreticalMean <= ($estimate+$delta))){ #is
                            covered by the CI
                                #print "adding␣good␣CI␣and␣good␣Length␣\n";
                                SaveIntoDatabase($estimate, $delta, $conf,
                                    $totalObs, $transObs, 1, 1);
                        } else { #is not covered by the CI
                                #print "Adding␣Bad␣CI␣and␣good␣Length␣\n";
                                SaveIntoDatabase($estimate, $delta, $conf,
                                    $totalObs, $transObs, 0, 1);
                        }
                        CalculateDeltaOfCoverage();
                } else {
                        $badLength += 1;
                }
                $output =~ m/RandomNumberState: (\d.*)/;
                $seed = $1;
        }
        my $updateQuery = $dbh->prepare("UPDATE␣`".$model."_results`␣SET␣`
            rejectedRuns`␣=␣?,␣`totalRuns`␣=␣?,␣`goodRuns`␣=␣?,␣`badRuns`␣=
            ␣?,␣`coverage`␣=␣?,␣`deltaCoverage`␣=␣?
␣␣␣␣␣␣␣␣WHERE␣(`model`␣=␣?␣AND␣`load`␣=␣?␣AND␣`analysisMethod`␣=␣?␣AND␣`
    transientMethod`␣=␣?)");
        $updateQuery->execute($badLength, $goodLenghtTotal, $goodLGoodCi,
            $goodLBadCi, $coverage, $deltaCoverage, $model, $load, "O_".
            $method, $transMethod);
        $updateQuery->finish();
        return;
}

#main function
if(@ARGV == 4){
        InitializeVariables();
        ConnectToDatabase();
        CreateDatabase();
        FirstRun();
        FirstCheckpoint();
        SequentialAnalysis();
        $dbh->disconnect();
        exit;
} else {
        print "bad␣syntax␣use␣-␣perl␣source.pl␣<model>␣<load>␣<method␣of␣
            output␣analysis>␣<trans␣method>\n";
```

```
        exit;
}
```

# Appendix E

# Tables of Results per Model

| Model | Load | SOAM | Transient M. | Short Runs | No. Runs | No. Bad CIs | Coverage | $\Delta_{z_{\frac{1}{2}}}$ | $\sigma(\bar{L})$ | $\bar{L}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| AR(1) | 0.5 | SA/HW | Schruben | 675 | 3795 | 200 | 0.947 | 0.00711 | 3219.231 | 8170.585 |
| AR(1) | 0.5 | SA/HW | CumulativeMeans | 417 | 2699 | 200 | 0.926 | 0.00988 | 2983.592 | 6516.655 |
| AR(1) | 0.6 | SA/HW | Schruben | 608 | 3570 | 200 | 0.944 | 0.00754 | 4893.289 | 12033.640 |
| AR(1) | 0.6 | SA/HW | CumulativeMeans | 441 | 2751 | 200 | 0.927 | 0.00970 | 4602.059 | 10053.165 |
| AR(1) | 0.7 | SA/HW | Schruben | 448 | 2956 | 200 | 0.932 | 0.00905 | 8747.356 | 20262.207 |
| AR(1) | 0.7 | SA/HW | CumulativeMeans | 398 | 2667 | 200 | 0.925 | 0.01000 | 8318.180 | 17715.235 |
| AR(1) | 0.8 | SA/HW | Schruben | 566 | 3378 | 200 | 0.941 | 0.00796 | 18767.171 | 42372.108 |
| AR(1) | 0.8 | SA/HW | CumulativeMeans | 503 | 3033 | 200 | 0.934 | 0.00883 | 18483.988 | 39431.204 |
| AR(1) | 0.9 | SA/HW | Schruben | 474 | 3036 | 200 | 0.934 | 0.00882 | 72681.411 | 155621.993 |
| AR(1) | 0.9 | SA/HW | CumulativeMeans | 452 | 2679 | 202 | 0.925 | 0.01000 | 72567.916 | 153933.573 |
| AR(1) | 0.95 | SA/HW | CumulativeMeans | 1072 | 5934 | 200 | 0.966 | 0.00459 | 1008.319 | 3231.425 |
| AR(1) | 0.95 | SA/HW | Schruben | 1214 | 5819 | 200 | 0.966 | 0.00468 | 1143.345 | 3982.917 |
| AR(1) | 0.5 | DPBM(k=50) | Schruben | 584 | 3744 | 200 | 0.947 | 0.00720 | 243.942 | 1691.676 |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| AR(1) | 0.5 | DPBM(k=50) | CumulativeMeans | 666 | 3751 | 200 | 0.947 | 0.00719 | 240.757 | 1476.700 |
| AR(1) | 0.6 | DPBM(k=50) | CumulativeMeans | 599 | 3337 | 200 | 0.940 | 0.00805 | 235.852 | 1439.838 |
| AR(1) | 0.6 | DPBM(k=50) | Schruben | 538 | 3327 | 200 | 0.940 | 0.00808 | 242.533 | 1641.858 |
| AR(1) | 0.7 | DPBM(k=50) | CumulativeMeans | 595 | 2751 | 200 | 0.927 | 0.00970 | 216.761 | 1379.789 |
| AR(1) | 0.7 | DPBM(k=50) | Schruben | 432 | 2876 | 200 | 0.930 | 0.00930 | 255.833 | 1550.156 |
| AR(1) | 0.8 | DPBM(k=50) | CumulativeMeans | 156 | 5070 | 793 | 0.844 | 0.01000 | 156.161 | 1257.436 |
| AR(1) | 0.8 | DPBM(k=50) | Schruben | 726 | 3968 | 464 | 0.883 | 0.01000 | 349.033 | 1224.055 |
| AR(1) | 0.9 | DPBM(k=50) | Schruben | 704 | 9303 | 3828 | 0.589 | 0.01000 | 131.881 | 671.772 |
| AR(1) | 0.9 | DPBM(k=50) | CumulativeMeans | 1356 | 9228 | 3700 | 0.599 | 0.01000 | 38.465 | 1217.382 |
| AR(1) | 0.95 | DPBM(k=50) | Schruben | 586 | 9366 | 5421 | 0.421 | 0.01000 | 152.794 | 733.048 |
| AR(1) | 0.95 | DPBM(k=50) | CumulativeMeans | 1591 | 9468 | 5298 | 0.440 | 0.01000 | 45.812 | 1306.443 |
| AR(1) | 0.5 | DPBM(k=15) | CumulativeMeans | 807 | 3598 | 200 | 0.944 | 0.00749 | 404.457 | 1557.418 |
| AR(1) | 0.5 | DPBM(k=15) | Schruben | 797 | 3869 | 200 | 0.948 | 0.00698 | 414.672 | 1772.412 |
| AR(1) | 0.6 | DPBM(k=15) | CumulativeMeans | 855 | 3660 | 200 | 0.945 | 0.00736 | 400.568 | 1564.869 |
| AR(1) | 0.6 | DPBM(k=15) | Schruben | 695 | 3722 | 200 | 0.946 | 0.00724 | 416.280 | 1759.768 |
| AR(1) | 0.7 | DPBM(k=15) | CumulativeMeans | 816 | 3442 | 200 | 0.942 | 0.00782 | 394.017 | 1568.150 |
| AR(1) | 0.7 | DPBM(k=15) | Schruben | 571 | 3365 | 200 | 0.941 | 0.00799 | 423.858 | 1748.707 |
| AR(1) | 0.8 | DPBM(k=15) | CumulativeMeans | 705 | 3075 | 200 | 0.935 | 0.00872 | 390.444 | 1571.668 |
| AR(1) | 0.8 | DPBM(k=15) | Schruben | 504 | 3107 | 200 | 0.936 | 0.00863 | 434.167 | 1717.864 |
| AR(1) | 0.9 | DPBM(k=15) | Schruben | 1007 | 3610 | 379 | 0.895 | 0.01000 | 575.205 | 1366.529 |
| AR(1) | 0.9 | DPBM(k=15) | CumulativeMeans | 2 | 5861 | 1101 | 0.812 | 0.01000 | 314.575 | 1440.952 |
| AR(1) | 0.95 | DPBM(k=15) | Schruben | 1 | 9431 | 5350 | 0.433 | 0.01000 | 367.471 | 736.742 |
| AR(1) | 0.95 | DPBM(k=15) | CumulativeMeans | 86 | 9490 | 5264 | 0.445 | 0.01000 | 105.005 | 1324.826 |
| AR(1) | 0.5 | MSE-DPBM | CumulativeMeans | 839 | 3632 | 200 | 0.945 | 0.00742 | 402.417 | 1555.782 |
| AR(1) | 0.5 | MSE-DPBM | Schruben | 801 | 3865 | 200 | 0.948 | 0.00698 | 414.725 | 1772.796 |
| AR(1) | 0.6 | MSE-DPBM | CumulativeMeans | 853 | 3662 | 200 | 0.945 | 0.00736 | 400.543 | 1564.893 |
| AR(1) | 0.6 | MSE-DPBM | Schruben | 693 | 3738 | 200 | 0.946 | 0.00721 | 416.130 | 1760.648 |
| AR(1) | 0.7 | MSE-DPBM | CumulativeMeans | 814 | 3444 | 200 | 0.942 | 0.00781 | 393.882 | 1568.276 |
| AR(1) | 0.7 | MSE-DPBM | Schruben | 571 | 3365 | 200 | 0.941 | 0.00799 | 423.839 | 1748.730 |
| AR(1) | 0.8 | MSE-DPBM | CumulativeMeans | 705 | 3075 | 200 | 0.935 | 0.00872 | 390.444 | 1571.668 |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| AR(1) | 0.8 | MSE-DPBM | Schruben | 504 | 3107 | 200 | 0.936 | 0.00863 | 434.125 | 1717.915 |
| AR(1) | 0.9 | MSE-DPBM | Schruben | 1007 | 3610 | 379 | 0.895 | 0.01000 | 575.205 | 1366.529 |
| AR(1) | 0.9 | MSE-DPBM | CumulativeMeans | 2 | 5861 | 1101 | 0.812 | 0.01000 | 314.575 | 1440.952 |
| AR(1) | 0.95 | MSE-DPBM | Schruben | 1 | 9431 | 5350 | 0.433 | 0.01000 | 367.471 | 736.742 |
| AR(1) | 0.95 | MSE-DPBM | CumulativeMeans | 86 | 9490 | 5264 | 0.445 | 0.01000 | 105.005 | 1324.826 |
| AR(1) | 0.5 | Mod. MSE-DPBM | CumulativeMeans | 747 | 2686 | 200 | 0.926 | 0.00993 | 44.182 | 4150.951 |
| AR(1) | 0.5 | Mod. MSE-DPBM | Schruben | 108 | 3264 | 200 | 0.939 | 0.00823 | 23.447 | 4367.735 |
| AR(1) | 0.6 | Mod. MSE-DPBM | CumulativeMeans | 729 | 3039 | 200 | 0.934 | 0.00882 | 46.350 | 4169.857 |
| AR(1) | 0.6 | Mod. MSE-DPBM | Schruben | 132 | 3143 | 200 | 0.936 | 0.00853 | 26.415 | 4376.474 |
| AR(1) | 0.7 | Mod. MSE-DPBM | CumulativeMeans | 471 | 2987 | 200 | 0.933 | 0.00896 | 44.146 | 4199.055 |
| AR(1) | 0.7 | Mod. MSE-DPBM | Schruben | 173 | 3079 | 200 | 0.935 | 0.00871 | 30.011 | 4389.873 |
| AR(1) | 0.8 | Mod. MSE-DPBM | Schruben | 254 | 2906 | 200 | 0.931 | 0.00920 | 36.606 | 4414.145 |
| AR(1) | 0.8 | Mod. MSE-DPBM | CumulativeMeans | 348 | 2835 | 200 | 0.929 | 0.00943 | 39.056 | 4240.387 |
| AR(1) | 0.9 | Mod. MSE-DPBM | Schruben | 305 | 2673 | 201 | 0.925 | 0.01000 | 66.061 | 4477.811 |
| AR(1) | 0.9 | Mod. MSE-DPBM | CumulativeMeans | 335 | 2725 | 200 | 0.927 | 0.00979 | 42.040 | 4309.595 |
| AR(1) | 0.95 | Mod. MSE-DPBM | CumulativeMeans | 391 | 2766 | 200 | 0.928 | 0.00965 | 53.902 | 4398.186 |
| AR(1) | 0.95 | Mod. MSE-DPBM | Schruben | 189 | 2741 | 200 | 0.927 | 0.00974 | 156.266 | 4619.314 |

Table E.1: AR(1) Results Table

| Model | Load | SOAM | Transient M. | Short Runs | No. Runs | No. Bad CIs | Coverage | $\Delta_{z_{\frac{1}{2}}}$ | $\sigma(\bar{L})$ | $\bar{L}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| M/M/1 | 0.5 | SA/HW | Schruben | 647 | 4006 | 200 | 0.950 | 0.00674 | 7208.586 | 16168.487 |
| M/M/1 | 0.5 | SA/HW | CumulativeMeans | 441 | 2936 | 200 | 0.932 | 0.00911 | 6984.198 | 13166.879 |
| M/M/1 | 0.6 | SA/HW | Schruben | 580 | 3493 | 200 | 0.943 | 0.00770 | 12616.678 | 27014.707 |
| M/M/1 | 0.6 | SA/HW | CumulativeMeans | 373 | 2757 | 200 | 0.927 | 0.00968 | 12635.658 | 23166.342 |
| M/M/1 | 0.7 | SA/HW | Schruben | 477 | 2976 | 200 | 0.933 | 0.00900 | 24943.584 | 51868.517 |
| M/M/1 | 0.7 | SA/HW | CumulativeMeans | 434 | 2797 | 221 | 0.921 | 0.01000 | 24395.357 | 45923.890 |
| M/M/1 | 0.8 | SA/HW | Schruben | 423 | 2760 | 215 | 0.922 | 0.01000 | 63237.041 | 125773.314 |
| M/M/1 | 0.8 | SA/HW | CumulativeMeans | 498 | 2909 | 240 | 0.917 | 0.01000 | 63400.331 | 121014.307 |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| M/M/1 | 0.9 | SA/HW | Schruben | 418 | 2981 | 200 | 0.933 | 0.00898 | 291112.111 | 535423.103 |
| M/M/1 | 0.9 | SA/HW | CumulativeMeans | 414 | 2857 | 200 | 0.930 | 0.00936 | 286357.789 | 526884.095 |
| M/M/1 | 0.95 | SA/HW | Schruben | 405 | 2748 | 213 | 0.922 | 0.01000 | 1200026.279 | 2184140.667 |
| M/M/1 | 0.95 | SA/HW | CumulativeMeans | 410 | 2851 | 230 | 0.919 | 0.01000 | 1208585.396 | 2188953.118 |
| M/M/1 | 0.5 | DPBM(k=50) | Schruben | 725 | 4564 | 200 | 0.956 | 0.00594 | 3293.983 | 13188.417 |
| M/M/1 | 0.5 | DPBM(k=50) | CumulativeMeans | 732 | 4672 | 200 | 0.957 | 0.00580 | 3330.873 | 12889.818 |
| M/M/1 | 0.6 | DPBM(k=50) | Schruben | 775 | 4758 | 200 | 0.958 | 0.00570 | 5831.210 | 22970.244 |
| M/M/1 | 0.6 | DPBM(k=50) | CumulativeMeans | 870 | 5177 | 200 | 0.961 | 0.00525 | 5850.914 | 22628.886 |
| M/M/1 | 0.7 | DPBM(k=50) | Schruben | 688 | 4029 | 200 | 0.950 | 0.00671 | 11787.731 | 45475.396 |
| M/M/1 | 0.7 | DPBM(k=50) | CumulativeMeans | 749 | 4354 | 200 | 0.954 | 0.00622 | 11685.815 | 45002.962 |
| M/M/1 | 0.8 | DPBM(k=50) | Schruben | 617 | 3744 | 200 | 0.947 | 0.00720 | 31141.474 | 114391.614 |
| M/M/1 | 0.8 | DPBM(k=50) | CumulativeMeans | 615 | 3661 | 200 | 0.945 | 0.00736 | 31064.851 | 113756.911 |
| M/M/1 | 0.9 | DPBM(k=50) | CumulativeMeans | 437 | 3790 | 200 | 0.947 | 0.00712 | 155804.302 | 497605.707 |
| M/M/1 | 0.9 | DPBM(k=50) | Schruben | 479 | 4009 | 200 | 0.950 | 0.00674 | 151417.284 | 502411.503 |
| M/M/1 | 0.95 | DPBM(k=50) | CumulativeMeans | 563 | 2685 | 200 | 0.926 | 0.00993 | 957499.138 | 1772073.000 |
| M/M/1 | 0.95 | DPBM(k=50) | Schruben | 386 | 2874 | 200 | 0.930 | 0.00930 | 870420.577 | 1888071.570 |
| M/M/1 | 0.5 | DPBM(k=15) | CumulativeMeans | 633 | 3742 | 200 | 0.947 | 0.00721 | 4434.984 | 13131.513 |
| M/M/1 | 0.5 | DPBM(k=15) | Schruben | 704 | 4024 | 200 | 0.950 | 0.00671 | 4429.561 | 13404.649 |
| M/M/1 | 0.6 | DPBM(k=15) | CumulativeMeans | 497 | 3403 | 200 | 0.941 | 0.00790 | 7839.570 | 22987.054 |
| M/M/1 | 0.6 | DPBM(k=15) | Schruben | 547 | 3818 | 200 | 0.948 | 0.00707 | 7766.950 | 23334.039 |
| M/M/1 | 0.7 | DPBM(k=15) | Schruben | 442 | 3452 | 200 | 0.942 | 0.00779 | 15658.312 | 46622.590 |
| M/M/1 | 0.7 | DPBM(k=15) | CumulativeMeans | 425 | 3317 | 200 | 0.940 | 0.00810 | 15797.079 | 46339.172 |
| M/M/1 | 0.8 | DPBM(k=15) | CumulativeMeans | 531 | 3003 | 200 | 0.933 | 0.00892 | 40200.264 | 116946.390 |
| M/M/1 | 0.8 | DPBM(k=15) | Schruben | 554 | 3128 | 200 | 0.936 | 0.00857 | 40463.876 | 117964.435 |
| M/M/1 | 0.9 | DPBM(k=15) | CumulativeMeans | 466 | 2983 | 200 | 0.933 | 0.00898 | 221249.102 | 484340.515 |
| M/M/1 | 0.9 | DPBM(k=15) | Schruben | 576 | 3528 | 200 | 0.943 | 0.00763 | 193101.902 | 511076.351 |
| M/M/1 | 0.95 | DPBM(k=15) | CumulativeMeans | 1046 | 2920 | 242 | 0.917 | 0.01000 | 1132624.203 | 1582889.674 |
| M/M/1 | 0.95 | DPBM(k=15) | Schruben | 437 | 2821 | 225 | 0.920 | 0.01000 | 973657.420 | 1873689.061 |
| M/M/1 | 0.5 | MSE-DPBM | CumulativeMeans | 839 | 4021 | 200 | 0.950 | 0.00672 | 4409.576 | 13130.874 |
| M/M/1 | 0.5 | MSE-DPBM | Schruben | 700 | 3947 | 200 | 0.949 | 0.00684 | 4414.412 | 13407.614 |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| M/M/1 | 0.6 | MSE-DPBM | CumulativeMeans | 535 | 3669 | 200 | 0.945 | 0.00735 | 7817.855 | 23009.636 |
| M/M/1 | 0.6 | MSE-DPBM | Schruben | 562 | 3894 | 200 | 0.949 | 0.00693 | 7703.258 | 23331.484 |
| M/M/1 | 0.7 | MSE-DPBM | Schruben | 517 | 3478 | 200 | 0.942 | 0.00774 | 15402.717 | 46621.981 |
| M/M/1 | 0.7 | MSE-DPBM | CumulativeMeans | 528 | 3509 | 200 | 0.943 | 0.00767 | 15553.193 | 46342.798 |
| M/M/1 | 0.8 | MSE-DPBM | CumulativeMeans | 539 | 3138 | 200 | 0.936 | 0.00855 | 39680.113 | 116769.455 |
| M/M/1 | 0.8 | MSE-DPBM | Schruben | 526 | 3139 | 200 | 0.936 | 0.00854 | 39836.457 | 117710.300 |
| M/M/1 | 0.9 | MSE-DPBM | CumulativeMeans | 363 | 2817 | 200 | 0.929 | 0.00948 | 219474.191 | 484102.308 |
| M/M/1 | 0.9 | MSE-DPBM | Schruben | 574 | 3579 | 200 | 0.944 | 0.00753 | 190757.628 | 511137.642 |
| M/M/1 | 0.95 | MSE-DPBM | CumulativeMeans | 1044 | 2915 | 241 | 0.917 | 0.01000 | 1121150.102 | 1578783.920 |
| M/M/1 | 0.95 | MSE-DPBM | Schruben | 429 | 2729 | 210 | 0.923 | 0.01000 | 966762.428 | 1869275.374 |
| M/M/1 | 0.5 | Mod. MSE-DPBM | CumulativeMeans | 808 | 3866 | 200 | 0.948 | 0.00698 | 4256.167 | 13017.297 |
| M/M/1 | 0.5 | Mod. MSE-DPBM | Schruben | 857 | 4267 | 200 | 0.953 | 0.00634 | 4231.421 | 13274.664 |
| M/M/1 | 0.6 | Mod. MSE-DPBM | CumulativeMeans | 507 | 3496 | 200 | 0.943 | 0.00770 | 7344.981 | 22682.353 |
| M/M/1 | 0.6 | Mod. MSE-DPBM | Schruben | 658 | 4112 | 200 | 0.951 | 0.00657 | 7253.206 | 23041.541 |
| M/M/1 | 0.7 | Mod. MSE-DPBM | Schruben | 490 | 3366 | 200 | 0.941 | 0.00799 | 14545.610 | 45840.963 |
| M/M/1 | 0.7 | Mod. MSE-DPBM | CumulativeMeans | 500 | 3453 | 200 | 0.942 | 0.00779 | 14652.814 | 45672.610 |
| M/M/1 | 0.8 | Mod. MSE-DPBM | CumulativeMeans | 604 | 3260 | 200 | 0.939 | 0.00824 | 36977.622 | 115407.086 |
| M/M/1 | 0.8 | Mod. MSE-DPBM | Schruben | 621 | 3346 | 200 | 0.940 | 0.00803 | 36945.963 | 116074.838 |
| M/M/1 | 0.9 | Mod. MSE-DPBM | CumulativeMeans | 715 | 4096 | 200 | 0.951 | 0.00660 | 161335.373 | 510614.788 |
| M/M/1 | 0.9 | Mod. MSE-DPBM | Schruben | 724 | 4138 | 200 | 0.952 | 0.00653 | 160673.174 | 510767.003 |
| M/M/1 | 0.95 | Mod. MSE-DPBM | Schruben | 516 | 3645 | 200 | 0.945 | 0.00739 | 674549.187 | 2091911.295 |
| M/M/1 | 0.95 | Mod. MSE-DPBM | CumulativeMeans | 510 | 3687 | 200 | 0.946 | 0.00731 | 678515.146 | 2108415.18 |

Table E.2: M/M/1 Results Table

| Model | Load | SOAM | Transient M. | Short Runs | No. Runs | No. Bad CIs | Coverage | $\Delta_{z\frac{1}{2}}$ | $\sigma(\bar{L})$ | $\bar{L}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| M/D/1 | 0.5 | SA/HW | Schruben | 650 | 3631 | 200 | 0.945 | 0.00742 | 1486.006 | 3508.210 |
| M/D/1 | 0.5 | SA/HW | CumulativeMeans | 464 | 3172 | 288 | 0.909 | 0.01000 | 1262.804 | 1973.141 |
| M/D/1 | 0.6 | SA/HW | Schruben | 488 | 3688 | 200 | 0.946 | 0.00731 | 3167.599 | 6561.700 |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| M/D/1 | 0.6 | SA/HW | CumulativeMeans | 445 | 3103 | 275 | 0.911 | 0.01000 | 2894.693 | 4546.648 |
| M/D/1 | 0.7 | SA/HW | Schruben | 400 | 2855 | 200 | 0.930 | 0.00936 | 7255.593 | 14458.083 |
| M/D/1 | 0.7 | SA/HW | CumulativeMeans | 375 | 3000 | 256 | 0.915 | 0.01000 | 6918.226 | 11361.186 |
| M/D/1 | 0.8 | SA/HW | Schruben | 447 | 2990 | 200 | 0.933 | 0.00896 | 21049.368 | 41184.397 |
| M/D/1 | 0.8 | SA/HW | CumulativeMeans | 366 | 2698 | 205 | 0.924 | 0.01000 | 20609.424 | 36071.729 |
| M/D/1 | 0.9 | SA/HW | Schruben | 406 | 2698 | 200 | 0.926 | 0.00989 | 111401.877 | 213543.009 |
| M/D/1 | 0.9 | SA/HW | CumulativeMeans | 438 | 2768 | 200 | 0.928 | 0.00965 | 111922.580 | 208702.591 |
| M/D/1 | 0.95 | SA/HW | Schruben | 448 | 3035 | 200 | 0.934 | 0.00883 | 525311.870 | 994545.019 |
| M/D/1 | 0.95 | SA/HW | CumulativeMeans | 422 | 2860 | 200 | 0.930 | 0.00935 | 526499.593 | 984025.428 |
| M/D/1 | 0.5 | DPBM(k=50) | Schruben | 523 | 3138 | 200 | 0.936 | 0.00855 | 792.459 | 1918.795 |
| M/D/1 | 0.5 | DPBM(k=50) | CumulativeMeans | 1073 | 4154 | 200 | 0.952 | 0.00651 | 721.699 | 1723.519 |
| M/D/1 | 0.6 | DPBM(k=50) | CumulativeMeans | 632 | 3765 | 200 | 0.947 | 0.00716 | 1630.169 | 4015.720 |
| M/D/1 | 0.6 | DPBM(k=50) | Schruben | 565 | 3561 | 200 | 0.944 | 0.00756 | 1621.651 | 4280.669 |
| M/D/1 | 0.7 | DPBM(k=50) | CumulativeMeans | 609 | 3864 | 200 | 0.948 | 0.00699 | 3865.489 | 10470.801 |
| M/D/1 | 0.7 | DPBM(k=50) | Schruben | 703 | 4225 | 200 | 0.953 | 0.00640 | 3799.760 | 10771.656 |
| M/D/1 | 0.8 | DPBM(k=50) | CumulativeMeans | 829 | 5030 | 200 | 0.960 | 0.00540 | 11295.408 | 34572.847 |
| M/D/1 | 0.8 | DPBM(k=50) | Schruben | 858 | 4975 | 200 | 0.960 | 0.00546 | 11256.344 | 34973.221 |
| M/D/1 | 0.9 | DPBM(k=50) | CumulativeMeans | 627 | 4227 | 200 | 0.953 | 0.00640 | 60107.185 | 194673.975 |
| M/D/1 | 0.9 | DPBM(k=50) | Schruben | 689 | 4304 | 200 | 0.954 | 0.00629 | 58640.098 | 197600.655 |
| M/D/1 | 0.95 | DPBM(k=50) | CumulativeMeans | 402 | 3029 | 200 | 0.934 | 0.00884 | 379242.871 | 854543.868 |
| M/D/1 | 0.95 | DPBM(k=50) | Schruben | 322 | 3256 | 200 | 0.939 | 0.00825 | 346136.258 | 887291.218 |
| M/D/1 | 0.5 | DPBM(k=15) | Schruben | 421 | 2679 | 202 | 0.925 | 0.01000 | 913.979 | 2083.976 |
| M/D/1 | 0.5 | DPBM(k=15) | CumulativeMeans | 656 | 3326 | 200 | 0.940 | 0.00808 | 870.456 | 1870.811 |
| M/D/1 | 0.6 | DPBM(k=15) | CumulativeMeans | 562 | 3163 | 200 | 0.937 | 0.00848 | 1928.953 | 4275.320 |
| M/D/1 | 0.6 | DPBM(k=15) | Schruben | 520 | 3164 | 200 | 0.937 | 0.00848 | 1941.673 | 4543.456 |
| M/D/1 | 0.7 | DPBM(k=15) | CumulativeMeans | 401 | 2809 | 200 | 0.929 | 0.00951 | 4708.362 | 10955.683 |
| M/D/1 | 0.7 | DPBM(k=15) | Schruben | 441 | 2976 | 200 | 0.933 | 0.00900 | 4667.201 | 11242.866 |
| M/D/1 | 0.8 | DPBM(k=15) | Schruben | 683 | 3817 | 200 | 0.948 | 0.00707 | 13771.218 | 36106.431 |
| M/D/1 | 0.8 | DPBM(k=15) | CumulativeMeans | 636 | 3554 | 200 | 0.944 | 0.00758 | 13832.280 | 35514.696 |
| M/D/1 | 0.9 | DPBM(k=15) | CumulativeMeans | 396 | 3058 | 200 | 0.935 | 0.00876 | 79711.431 | 194515.089 |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| M/D/1 | 0.9 | DPBM(k=15) | Schruben | 484 | 3373 | 200 | 0.941 | 0.00797 | 75303.398 | 201841.318 |
| M/D/1 | 0.95 | DPBM(k=15) | CumulativeMeans | 632 | 2915 | 241 | 0.917 | 0.01000 | 483124.939 | 789174.818 |
| M/D/1 | 0.95 | DPBM(k=15) | Schruben | 345 | 2866 | 200 | 0.930 | 0.00933 | 401747.787 | 906234.555 |
| M/D/1 | 0.5 | MSE-DPBM | Schruben | 421 | 2679 | 202 | 0.925 | 0.01000 | 913.979 | 2083.976 |
| M/D/1 | 0.5 | MSE-DPBM | CumulativeMeans | 656 | 3326 | 200 | 0.940 | 0.00808 | 870.485 | 1870.917 |
| M/D/1 | 0.6 | MSE-DPBM | CumulativeMeans | 561 | 3164 | 200 | 0.937 | 0.00848 | 1928.434 | 4278.186 |
| M/D/1 | 0.6 | MSE-DPBM | Schruben | 523 | 3172 | 200 | 0.937 | 0.00846 | 1937.244 | 4542.505 |
| M/D/1 | 0.7 | MSE-DPBM | CumulativeMeans | 403 | 2827 | 200 | 0.929 | 0.00945 | 4710.115 | 10967.461 |
| M/D/1 | 0.7 | MSE-DPBM | Schruben | 444 | 3020 | 200 | 0.934 | 0.00887 | 4663.016 | 11230.304 |
| M/D/1 | 0.8 | MSE-DPBM | CumulativeMeans | 627 | 3531 | 200 | 0.943 | 0.00762 | 13755.733 | 35532.354 |
| M/D/1 | 0.8 | MSE-DPBM | Schruben | 666 | 3760 | 200 | 0.947 | 0.00717 | 13750.568 | 36074.302 |
| M/D/1 | 0.9 | MSE-DPBM | CumulativeMeans | 392 | 3016 | 200 | 0.934 | 0.00888 | 80038.832 | 195303.320 |
| M/D/1 | 0.9 | MSE-DPBM | Schruben | 508 | 3536 | 200 | 0.943 | 0.00761 | 74750.247 | 202021.853 |
| M/D/1 | 0.95 | MSE-DPBM | CumulativeMeans | 601 | 2839 | 228 | 0.920 | 0.01000 | 477897.638 | 788486.261 |
| M/D/1 | 0.95 | MSE-DPBM | Schruben | 354 | 2975 | 200 | 0.933 | 0.00900 | 397839.569 | 903085.391 |
| M/D/1 | 0.5 | Mod. MSE-DPBM | CumulativeMeans | 0 | 2983 | 253 | 0.915 | 0.01000 | 220.794 | 4126.707 |
| M/D/1 | 0.5 | Mod. MSE-DPBM | Schruben | 0 | 2920 | 242 | 0.917 | 0.01000 | 221.257 | 4408.905 |
| M/D/1 | 0.6 | Mod. MSE-DPBM | CumulativeMeans | 0 | 3156 | 285 | 0.910 | 0.01000 | 1498.751 | 5050.176 |
| M/D/1 | 0.6 | Mod. MSE-DPBM | Schruben | 0 | 2955 | 248 | 0.916 | 0.01000 | 1461.769 | 5332.624 |
| M/D/1 | 0.7 | Mod. MSE-DPBM | CumulativeMeans | 382 | 2837 | 200 | 0.930 | 0.00942 | 4557.025 | 10929.803 |
| M/D/1 | 0.7 | Mod. MSE-DPBM | Schruben | 520 | 3314 | 200 | 0.940 | 0.00811 | 4443.292 | 11238.517 |
| M/D/1 | 0.8 | Mod. MSE-DPBM | Schruben | 689 | 4029 | 200 | 0.950 | 0.00671 | 13141.179 | 35571.430 |
| M/D/1 | 0.8 | Mod. MSE-DPBM | CumulativeMeans | 709 | 3995 | 200 | 0.950 | 0.00676 | 13226.494 | 35170.016 |
| M/D/1 | 0.9 | Mod. MSE-DPBM | Schruben | 842 | 4122 | 200 | 0.951 | 0.00656 | 66837.778 | 200173.155 |
| M/D/1 | 0.9 | Mod. MSE-DPBM | CumulativeMeans | 887 | 4395 | 200 | 0.954 | 0.00616 | 67403.132 | 199315.195 |
| M/D/1 | 0.95 | Mod. MSE-DPBM | CumulativeMeans | 686 | 3821 | 200 | 0.948 | 0.00706 | 304427.570 | 938234.761 |
| M/D/1 | 0.95 | Mod. MSE-DPBM | Schruben | 615 | 3611 | 200 | 0.945 | 0.00746 | 306503.560 | 943886.910 |

Table E.3: M/D/1 Results Table

| Model | Load | SOAM | Transient M. | Short Runs | No. Runs | No. Bad CIs | Coverage | $\Delta_{z\frac{1}{2}}$ | $\sigma(\bar{L})$ | $\bar{L}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| M/$H_2$/1 | 0.5 | SA/HW | Schruben | 452 | 2997 | 200 | 0.933 | 0.00893 | 57674.039 | 123912.116 |
| M/$H_2$/1 | 0.5 | SA/HW | CumulativeMeans | 476 | 2885 | 200 | 0.931 | 0.00927 | 57175.554 | 122893.460 |
| M/$H_2$/1 | 0.6 | SA/HW | Schruben | 418 | 2835 | 200 | 0.929 | 0.00943 | 87043.047 | 181578.977 |
| M/$H_2$/1 | 0.6 | SA/HW | CumulativeMeans | 432 | 2720 | 200 | 0.926 | 0.00981 | 86317.419 | 182308.578 |
| M/$H_2$/1 | 0.7 | SA/HW | Schruben | 417 | 2731 | 200 | 0.927 | 0.00977 | 149843.794 | 301715.945 |
| M/$H_2$/1 | 0.7 | SA/HW | CumulativeMeans | 412 | 2766 | 200 | 0.928 | 0.00965 | 151528.966 | 305956.040 |
| M/$H_2$/1 | 0.8 | SA/HW | Schruben | 388 | 2834 | 200 | 0.929 | 0.00943 | 312137.848 | 605675.193 |
| M/$H_2$/1 | 0.8 | SA/HW | CumulativeMeans | 423 | 3006 | 200 | 0.933 | 0.00891 | 310540.479 | 612048.825 |
| M/$H_2$/1 | 0.9 | SA/HW | CumulativeMeans | 479 | 3072 | 200 | 0.935 | 0.00872 | 1053390.608 | 2067673.817 |
| M/$H_2$/1 | 0.9 | SA/HW | Schruben | 474 | 3034 | 200 | 0.934 | 0.00883 | 1060370.904 | 2059401.160 |
| M/$H_2$/1 | 0.95 | SA/HW | CumulativeMeans | 407 | 2736 | 211 | 0.923 | 0.01000 | 4079278.549 | 7715059.357 |
| M/$H_2$/1 | 0.95 | SA/HW | Schruben | 422 | 2757 | 200 | 0.927 | 0.00968 | 4031780.527 | 7711850.075 |
| M/$H_2$/1 | 0.5 | DPBM(k=50) | CumulativeMeans | 691 | 4248 | 200 | 0.953 | 0.00637 | 22913.856 | 119351.333 |
| M/$H_2$/1 | 0.5 | DPBM(k=50) | Schruben | 736 | 4436 | 200 | 0.955 | 0.00611 | 22762.455 | 119298.166 |
| M/$H_2$/1 | 0.6 | DPBM(k=50) | CumulativeMeans | 756 | 4450 | 200 | 0.955 | 0.00609 | 36942.403 | 176712.288 |
| M/$H_2$/1 | 0.6 | DPBM(k=50) | Schruben | 645 | 4082 | 200 | 0.951 | 0.00662 | 36773.776 | 176470.692 |
| M/$H_2$/1 | 0.7 | DPBM(k=50) | CumulativeMeans | 747 | 3906 | 200 | 0.949 | 0.00691 | 62344.172 | 290181.687 |
| M/$H_2$/1 | 0.7 | DPBM(k=50) | Schruben | 814 | 4131 | 200 | 0.952 | 0.00655 | 62800.599 | 289690.302 |
| M/$H_2$/1 | 0.8 | DPBM(k=50) | Schruben | 658 | 4488 | 200 | 0.955 | 0.00604 | 137660.641 | 572127.241 |
| M/$H_2$/1 | 0.8 | DPBM(k=50) | CumulativeMeans | 848 | 4536 | 200 | 0.956 | 0.00597 | 128089.001 | 578954.616 |
| M/$H_2$/1 | 0.9 | DPBM(k=50) | Schruben | 311 | 3062 | 200 | 0.935 | 0.00875 | 727465.175 | 1834404.883 |
| M/$H_2$/1 | 0.9 | DPBM(k=50) | CumulativeMeans | 292 | 3487 | 200 | 0.943 | 0.00772 | 602332.617 | 1928061.858 |
| M/$H_2$/1 | 0.95 | DPBM(k=50) | Schruben | 795 | 2794 | 200 | 0.928 | 0.00956 | 3443150.693 | 5756450.401 |
| M/$H_2$/1 | 0.95 | DPBM(k=50) | CumulativeMeans | 755 | 2949 | 200 | 0.932 | 0.00907 | 3299919.784 | 6028409.636 |
| M/$H_2$/1 | 0.5 | DPBM(k=15) | Schruben | 626 | 3573 | 200 | 0.944 | 0.00754 | 34424.218 | 119374.135 |
| M/$H_2$/1 | 0.5 | DPBM(k=15) | CumulativeMeans | 633 | 3621 | 200 | 0.945 | 0.00744 | 34284.229 | 119404.664 |
| M/$H_2$/1 | 0.6 | DPBM(k=15) | CumulativeMeans | 474 | 3485 | 200 | 0.943 | 0.00772 | 51824.921 | 177562.611 |
| M/$H_2$/1 | 0.6 | DPBM(k=15) | Schruben | 492 | 3561 | 200 | 0.944 | 0.00756 | 51577.711 | 177009.109 |
| M/$H_2$/1 | 0.7 | DPBM(k=15) | Schruben | 590 | 3639 | 200 | 0.945 | 0.00740 | 91350.152 | 294166.985 |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $M/H_2/1$ | 0.7 | DPBM(k=15) | CumulativeMeans | 559 | 3483 | 200 | 0.943 | 0.00773 | 90000.330 | 295151.194 |
| $M/H_2/1$ | 0.8 | DPBM(k=15) | Schruben | 572 | 3985 | 200 | 0.950 | 0.00678 | 189633.769 | 579184.330 |
| $M/H_2/1$ | 0.8 | DPBM(k=15) | CumulativeMeans | 601 | 3832 | 200 | 0.948 | 0.00704 | 181690.503 | 585626.484 |
| $M/H_2/1$ | 0.9 | DPBM(k=15) | Schruben | 373 | 2716 | 200 | 0.926 | 0.00982 | 879138.899 | 1815273.908 |
| $M/H_2/1$ | 0.9 | DPBM(k=15) | CumulativeMeans | 399 | 3002 | 200 | 0.933 | 0.00892 | 805121.742 | 1902458.539 |
| $M/H_2/1$ | 0.95 | DPBM(k=15) | Schruben | 912 | 2892 | 237 | 0.918 | 0.01000 | 3845532.033 | 5788191.605 |
| $M/H_2/1$ | 0.95 | DPBM(k=15) | CumulativeMeans | 846 | 2897 | 238 | 0.918 | 0.01000 | 3769066.259 | 6013705.553 |
| $M/H_2/1$ | 0.5 | MSE-DPBM | Schruben | 598 | 3535 | 200 | 0.943 | 0.00762 | 33506.748 | 119182.978 |
| $M/H_2/1$ | 0.5 | MSE-DPBM | CumulativeMeans | 592 | 3540 | 200 | 0.944 | 0.00761 | 33242.296 | 119161.481 |
| $M/H_2/1$ | 0.6 | MSE-DPBM | CumulativeMeans | 492 | 3584 | 200 | 0.944 | 0.00752 | 50584.745 | 177731.099 |
| $M/H_2/1$ | 0.6 | MSE-DPBM | Schruben | 515 | 3745 | 200 | 0.947 | 0.00720 | 49834.664 | 176970.053 |
| $M/H_2/1$ | 0.7 | MSE-DPBM | Schruben | 570 | 3572 | 200 | 0.944 | 0.00754 | 88976.751 | 293612.637 |
| $M/H_2/1$ | 0.7 | MSE-DPBM | CumulativeMeans | 636 | 3406 | 200 | 0.941 | 0.00790 | 88059.564 | 294792.345 |
| $M/H_2/1$ | 0.8 | MSE-DPBM | CumulativeMeans | 596 | 3837 | 200 | 0.948 | 0.00703 | 178609.463 | 585953.650 |
| $M/H_2/1$ | 0.8 | MSE-DPBM | Schruben | 694 | 4093 | 200 | 0.951 | 0.00660 | 185337.936 | 581580.049 |
| $M/H_2/1$ | 0.9 | MSE-DPBM | Schruben | 384 | 2818 | 200 | 0.929 | 0.00948 | 870720.886 | 1806992.219 |
| $M/H_2/1$ | 0.9 | MSE-DPBM | CumulativeMeans | 309 | 2869 | 200 | 0.930 | 0.00932 | 795247.946 | 1895587.158 |
| $M/H_2/1$ | 0.95 | MSE-DPBM | Schruben | 861 | 2717 | 208 | 0.923 | 0.01000 | 3806777.815 | 5786733.224 |
| $M/H_2/1$ | 0.95 | MSE-DPBM | CumulativeMeans | 813 | 2803 | 222 | 0.921 | 0.01000 | 3727494.191 | 6012581.196 |
| $M/H_2/1$ | 0.5 | Mod. MSE-DPBM | Schruben | 661 | 3664 | 200 | 0.945 | 0.00736 | 30720.824 | 118061.258 |
| $M/H_2/1$ | 0.5 | Mod. MSE-DPBM | CumulativeMeans | 656 | 3732 | 200 | 0.946 | 0.00723 | 30544.433 | 118893.504 |
| $M/H_2/1$ | 0.6 | Mod. MSE-DPBM | Schruben | 479 | 3781 | 200 | 0.947 | 0.00713 | 46094.609 | 175255.109 |
| $M/H_2/1$ | 0.6 | Mod. MSE-DPBM | CumulativeMeans | 438 | 3516 | 200 | 0.943 | 0.00766 | 45926.235 | 175854.039 |
| $M/H_2/1$ | 0.7 | Mod. MSE-DPBM | Schruben | 556 | 3568 | 200 | 0.944 | 0.00755 | 79815.794 | 290849.409 |
| $M/H_2/1$ | 0.7 | Mod. MSE-DPBM | CumulativeMeans | 528 | 3429 | 200 | 0.942 | 0.00784 | 78827.742 | 290876.727 |
| $M/H_2/1$ | 0.8 | Mod. MSE-DPBM | CumulativeMeans | 722 | 4142 | 200 | 0.952 | 0.00653 | 153530.095 | 581265.281 |
| $M/H_2/1$ | 0.8 | Mod. MSE-DPBM | Schruben | 648 | 4144 | 200 | 0.952 | 0.00653 | 156278.842 | 580833.611 |
| $M/H_2/1$ | 0.9 | Mod. MSE-DPBM | Schruben | 702 | 4174 | 200 | 0.952 | 0.00648 | 548093.616 | 1980632.137 |
| $M/H_2/1$ | 0.9 | Mod. MSE-DPBM | CumulativeMeans | 708 | 4367 | 200 | 0.954 | 0.00620 | 542823.044 | 1985828.711 |
| $M/H_2/1$ | 0.95 | Mod. MSE-DPBM | CumulativeMeans | 500 | 3529 | 200 | 0.943 | 0.00763 | 2226171.426 | 7115973.626 |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| M/$H_2$/1 | 0.95 | Mod. MSE-DPBM | Schruben | 509 | 3549 | 200 | 0.944 | 0.00759 | 2237366.458 | 7151828.159 |

Table E.4: M/$H_2$/1 Results Table

| Model | Load | SOAM | Transient M. | Short Runs | No. Runs | No. Bad CIs | Coverage | $\Delta_{z\frac{1}{2}}$ | $\sigma(\bar{L})$ | $\bar{L}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| QNet | 0.5 | SA/HW | Schruben | 508 | 3225 | 200 | 0.938 | 0.00832 | 5381.268 | 12504.138 |
| QNet | 0.5 | SA/HW | CumulativeMeans | 224 | 5266 | 350 | 0.934 | 0.00673 | 9179.315 | 17442.026 |
| QNet | 0.6 | SA/HW | Schruben | 564 | 3489 | 200 | 0.943 | 0.00771 | 8603.445 | 19470.975 |
| QNet | 0.6 | SA/HW | CumulativeMeans | 188 | 4888 | 338 | 0.931 | 0.00711 | 15006.602 | 27747.440 |
| QNet | 0.7 | SA/HW | Schruben | 441 | 3073 | 200 | 0.935 | 0.00872 | 15916.497 | 33833.185 |
| QNet | 0.7 | SA/HW | CumulativeMeans | 202 | 5110 | 328 | 0.936 | 0.00672 | 26535.484 | 48129.322 |
| QNet | 0.8 | SA/HW | Schruben | 479 | 2899 | 200 | 0.931 | 0.00923 | 35360.852 | 72506.196 |
| QNet | 0.8 | SA/HW | CumulativeMeans | 192 | 5648 | 348 | 0.938 | 0.00627 | 59114.796 | 101297.488 |
| QNet | 0.9 | SA/HW | Schruben | 479 | 3221 | 200 | 0.938 | 0.00833 | 130138.562 | 251708.501 |
| QNet | 0.9 | SA/HW | CumulativeMeans | 142 | 4934 | 336 | 0.932 | 0.00703 | 202153.624 | 325588.813 |
| QNet | 0.95 | SA/HW | Schruben | 471 | 3005 | 257 | 0.914 | 0.01000 | 515755.430 | 931039.710 |
| QNet | 0.95 | SA/HW | CumulativeMeans | 475 | 2969 | 200 | 0.933 | 0.00902 | 524835.146 | 1097713.116 |
| QNet | 0.5 | DPBM(k=50) | CumulativeMeans | 611 | 3521 | 200 | 0.943 | 0.00765 | 2198.007 | 10601.130 |
| QNet | 0.5 | DPBM(k=50) | Schruben | 630 | 3682 | 200 | 0.946 | 0.00732 | 2115.319 | 9635.670 |
| QNet | 0.6 | DPBM(k=50) | Schruben | 697 | 3934 | 200 | 0.949 | 0.00686 | 3586.181 | 15441.116 |
| QNet | 0.6 | DPBM(k=50) | CumulativeMeans | 646 | 3663 | 200 | 0.945 | 0.00736 | 3713.480 | 16958.936 |
| QNet | 0.7 | DPBM(k=50) | CumulativeMeans | 620 | 3646 | 200 | 0.945 | 0.00739 | 7054.980 | 30296.156 |
| QNet | 0.7 | DPBM(k=50) | Schruben | 589 | 3789 | 200 | 0.947 | 0.00712 | 6746.274 | 27699.527 |
| QNet | 0.8 | DPBM(k=50) | Schruben | 753 | 3717 | 200 | 0.946 | 0.00725 | 15919.411 | 62351.794 |
| QNet | 0.8 | DPBM(k=50) | CumulativeMeans | 752 | 3763 | 200 | 0.947 | 0.00717 | 16332.167 | 66918.458 |
| QNet | 0.9 | DPBM(k=50) | CumulativeMeans | 647 | 4176 | 200 | 0.952 | 0.00648 | 62485.341 | 234035.325 |
| QNet | 0.9 | DPBM(k=50) | Schruben | 739 | 4438 | 200 | 0.955 | 0.00610 | 61758.770 | 224066.710 |
| QNet | 0.95 | DPBM(k=50) | CumulativeMeans | 469 | 3513 | 200 | 0.943 | 0.00766 | 272752.366 | 863012.857 |
| QNet | 0.95 | DPBM(k=50) | Schruben | 455 | 3397 | 200 | 0.941 | 0.00792 | 270505.613 | 850299.943 |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| QNet | 0.5 | DPBM(k=15) | CumulativeMeans | 559 | 3261 | 200 | 0.939 | 0.00824 | 3169.202 | 11021.483 |
| QNet | 0.5 | DPBM(k=15) | Schruben | 558 | 3435 | 200 | 0.942 | 0.00783 | 2994.940 | 10029.950 |
| QNet | 0.6 | DPBM(k=15) | CumulativeMeans | 671 | 3508 | 200 | 0.943 | 0.00767 | 5349.467 | 18014.899 |
| QNet | 0.6 | DPBM(k=15) | Schruben | 717 | 3946 | 200 | 0.949 | 0.00684 | 5086.901 | 16372.203 |
| QNet | 0.7 | DPBM(k=15) | Schruben | 729 | 3721 | 200 | 0.946 | 0.00725 | 9405.755 | 29329.385 |
| QNet | 0.7 | DPBM(k=15) | CumulativeMeans | 691 | 3652 | 200 | 0.945 | 0.00738 | 9516.081 | 31998.582 |
| QNet | 0.8 | DPBM(k=15) | CumulativeMeans | 675 | 3446 | 200 | 0.942 | 0.00781 | 22528.595 | 71674.439 |
| QNet | 0.8 | DPBM(k=15) | Schruben | 682 | 3471 | 200 | 0.942 | 0.00775 | 22145.206 | 66203.273 |
| QNet | 0.9 | DPBM(k=15) | CumulativeMeans | 765 | 4041 | 200 | 0.951 | 0.00669 | 81591.996 | 249732.725 |
| QNet | 0.9 | DPBM(k=15) | Schruben | 822 | 4115 | 200 | 0.951 | 0.00657 | 80564.666 | 239103.012 |
| QNet | 0.95 | DPBM(k=15) | CumulativeMeans | 613 | 3347 | 200 | 0.940 | 0.00803 | 336205.897 | 914282.836 |
| QNet | 0.95 | DPBM(k=15) | Schruben | 535 | 2909 | 200 | 0.931 | 0.00920 | 335193.558 | 901398.591 |
| QNet | 0.5 | MSE-DPBM | CumulativeMeans | 560 | 3260 | 200 | 0.939 | 0.00824 | 3154.562 | 11014.078 |
| QNet | 0.5 | MSE-DPBM | Schruben | 558 | 3447 | 200 | 0.942 | 0.00780 | 2987.629 | 10038.749 |
| QNet | 0.6 | MSE-DPBM | CumulativeMeans | 667 | 3475 | 200 | 0.942 | 0.00774 | 5349.395 | 18009.959 |
| QNet | 0.6 | MSE-DPBM | Schruben | 717 | 3927 | 200 | 0.949 | 0.00688 | 5063.519 | 16368.316 |
| QNet | 0.7 | MSE-DPBM | Schruben | 721 | 3698 | 200 | 0.946 | 0.00729 | 9344.426 | 29246.952 |
| QNet | 0.7 | MSE-DPBM | CumulativeMeans | 682 | 3613 | 200 | 0.945 | 0.00746 | 9489.436 | 32037.522 |
| QNet | 0.8 | MSE-DPBM | CumulativeMeans | 675 | 3422 | 200 | 0.942 | 0.00786 | 22413.793 | 71660.471 |
| QNet | 0.8 | MSE-DPBM | Schruben | 680 | 3489 | 200 | 0.943 | 0.00771 | 21868.271 | 66022.300 |
| QNet | 0.9 | MSE-DPBM | CumulativeMeans | 763 | 4043 | 200 | 0.951 | 0.00668 | 81323.643 | 249584.312 |
| QNet | 0.9 | MSE-DPBM | Schruben | 815 | 4122 | 200 | 0.951 | 0.00656 | 79809.554 | 238995.849 |
| QNet | 0.95 | MSE-DPBM | Schruben | 500 | 2843 | 200 | 0.930 | 0.00940 | 331072.697 | 900649.050 |
| QNet | 0.95 | MSE-DPBM | CumulativeMeans | 625 | 3398 | 200 | 0.941 | 0.00791 | 334472.527 | 914983.845 |
| QNet | 0.5 | Mod. MSE-DPBM | CumulativeMeans | 583 | 3275 | 200 | 0.939 | 0.00820 | 3013.452 | 10923.571 |
| QNet | 0.5 | Mod. MSE-DPBM | Schruben | 543 | 3419 | 200 | 0.942 | 0.00787 | 2884.290 | 9956.261 |
| QNet | 0.6 | Mod. MSE-DPBM | CumulativeMeans | 640 | 3353 | 200 | 0.940 | 0.00802 | 4917.493 | 17635.350 |
| QNet | 0.6 | Mod. MSE-DPBM | Schruben | 697 | 3943 | 200 | 0.949 | 0.00685 | 4820.327 | 16105.243 |
| QNet | 0.7 | Mod. MSE-DPBM | Schruben | 688 | 3690 | 200 | 0.946 | 0.00731 | 8893.976 | 28723.943 |
| QNet | 0.7 | Mod. MSE-DPBM | CumulativeMeans | 625 | 3470 | 200 | 0.942 | 0.00775 | 8981.217 | 31359.621 |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| QNet | 0.8 | Mod. MSE-DPBM | Schruben | 679 | 3284 | 200 | 0.939 | 0.00818 | 20441.343 | 63807.471 |
| QNet | 0.8 | Mod. MSE-DPBM | CumulativeMeans | 637 | 3357 | 200 | 0.940 | 0.00801 | 20789.104 | 68852.867 |
| QNet | 0.9 | Mod. MSE-DPBM | CumulativeMeans | 568 | 3380 | 200 | 0.941 | 0.00795 | 76787.854 | 228340.124 |
| QNet | 0.9 | Mod. MSE-DPBM | Schruben | 578 | 3322 | 200 | 0.940 | 0.00809 | 78143.262 | 222201.519 |
| QNet | 0.95 | Mod. MSE-DPBM | CumulativeMeans | 425 | 2915 | 241 | 0.917 | 0.01000 | 329586.959 | 810292.465 |
| QNet | 0.95 | Mod. MSE-DPBM | Schruben | 457 | 3005 | 257 | 0.914 | 0.01000 | 332848.919 | 796225.680 |

Table E.5: QNet Results Table

| Model | Load | SOAM | Transient M. | Batch Size $m$ | No. of Batces $k$ | Optimal Batch Size $m^*$ |
|---|---|---|---|---|---|---|
| AR(1) | 0.5 | SA/HW | Schruben | 17.0792 | 153.483 | 0 |
| AR(1) | 0.5 | SA/HW | CumulativeMeans | 13.5929 | 143.929 | 0 |
| AR(1) | 0.6 | SA/HW | Schruben | 17.2308 | 154.344 | 0 |
| AR(1) | 0.6 | SA/HW | CumulativeMeans | 14.2279 | 144.677 | 0 |
| AR(1) | 0.7 | SA/HW | Schruben | 18.2316 | 151.037 | 0 |
| AR(1) | 0.7 | SA/HW | CumulativeMeans | 15.1788 | 144.602 | 0 |
| AR(1) | 0.8 | SA/HW | Schruben | 19.1383 | 146.508 | 0 |
| AR(1) | 0.8 | SA/HW | CumulativeMeans | 16.2864 | 144.649 | 0 |
| AR(1) | 0.9 | SA/HW | Schruben | 21.2128 | 142.427 | 0 |
| AR(1) | 0.9 | SA/HW | CumulativeMeans | 18.2909 | 143.434 | 0 |
| AR(1) | 0.95 | SA/HW | Schruben | 25.1023 | 142.105 | 0 |
| AR(1) | 0.95 | SA/HW | CumulativeMeans | 20.9239 | 146.654 | 0 |
| $M/H_2/1$ | 0.5 | SA/HW | Schruben | 888.112 | 144.25 | 0 |
| $M/H_2/1$ | 0.5 | SA/HW | CumulativeMeans | 881.578 | 144.068 | 0 |
| $M/H_2/1$ | 0.6 | SA/HW | Schruben | 1316.18 | 143.063 | 0 |
| $M/H_2/1$ | 0.6 | SA/HW | CumulativeMeans | 1311.41 | 144.401 | 0 |
| $M/H_2/1$ | 0.7 | SA/HW | Schruben | 2153.26 | 144.464 | 0 |
| $M/H_2/1$ | 0.7 | SA/HW | CumulativeMeans | 2188.18 | 144.5 | 0 |
| $M/H_2/1$ | 0.8 | SA/HW | Schruben | 4333.67 | 144.721 | 0 |
| $M/H_2/1$ | 0.8 | SA/HW | CumulativeMeans | 4391.85 | 143.855 | 0 |

| | | | | | | |
|---|---|---|---|---|---|---|
| M/$H_2$/1 | 0.9 | SA/HW | Schruben | 14923.7 | 143.694 | 0 |
| M/$H_2$/1 | 0.9 | SA/HW | CumulativeMeans | 14973.9 | 144.028 | 0 |
| M/$H_2$/1 | 0.95 | SA/HW | Schruben | 54640.8 | 145.627 | 0 |
| M/$H_2$/1 | 0.95 | SA/HW | CumulativeMeans | 54508.2 | 145.398 | 0 |
| M/D/1 | 0.5 | SA/HW | Schruben | 22.9424 | 149.987 | 0 |
| M/D/1 | 0.5 | SA/HW | CumulativeMeans | 13.9227 | 145.388 | 0 |
| M/D/1 | 0.6 | SA/HW | Schruben | 45.1456 | 145.399 | 0 |
| M/D/1 | 0.6 | SA/HW | CumulativeMeans | 31.9914 | 145.362 | 0 |
| M/D/1 | 0.7 | SA/HW | Schruben | 101.899 | 145.009 | 0 |
| M/D/1 | 0.7 | SA/HW | CumulativeMeans | 81.1836 | 145.613 | 0 |
| M/D/1 | 0.8 | SA/HW | Schruben | 295.178 | 144.632 | 0 |
| M/D/1 | 0.8 | SA/HW | CumulativeMeans | 259.606 | 145.617 | 0 |
| M/D/1 | 0.9 | SA/HW | Schruben | 1536.63 | 144.759 | 0 |
| M/D/1 | 0.9 | SA/HW | CumulativeMeans | 1495.42 | 144.999 | 0 |
| M/D/1 | 0.95 | SA/HW | Schruben | 7182.94 | 144.827 | 0 |
| M/D/1 | 0.95 | SA/HW | CumulativeMeans | 7147.95 | 144.74 | 0 |
| M/M/1 | 0.5 | SA/HW | Schruben | 114.564 | 143.693 | 0 |
| M/M/1 | 0.5 | SA/HW | CumulativeMeans | 94.605 | 144.841 | 0 |
| M/M/1 | 0.6 | SA/HW | Schruben | 190.3 | 144.608 | 0 |
| M/M/1 | 0.6 | SA/HW | CumulativeMeans | 166.518 | 144.348 | 0 |
| M/M/1 | 0.7 | SA/HW | Schruben | 372.142 | 143.47 | 0 |
| M/M/1 | 0.7 | SA/HW | CumulativeMeans | 336.701 | 143.805 | 0 |
| M/M/1 | 0.8 | SA/HW | Schruben | 900.564 | 143.529 | 0 |
| M/M/1 | 0.8 | SA/HW | CumulativeMeans | 862.321 | 143.969 | 0 |
| M/M/1 | 0.9 | SA/HW | Schruben | 3808.96 | 145.18 | 0 |
| M/M/1 | 0.9 | SA/HW | CumulativeMeans | 3755.58 | 144.943 | 0 |
| M/M/1 | 0.95 | SA/HW | Schruben | 15767.1 | 144.081 | 0 |
| M/M/1 | 0.95 | SA/HW | CumulativeMeans | 15776.4 | 144.051 | 0 |
| QNet | 0.5 | SA/HW | Schruben | 87.9731 | 144.192 | 0 |
| QNet | 0.5 | SA/HW | CumulativeMeans | 114.314 | 144.018 | 0 |

| | | | | | | |
|---|---|---|---|---|---|---|
| QNet | 0.6 | SA/HW | Schruben | 138.031 | 144.144 | 0 |
| QNet | 0.6 | SA/HW | CumulativeMeans | 185.091 | 144.072 | 0 |
| QNet | 0.7 | SA/HW | Schruben | 240.273 | 144.805 | 0 |
| QNet | 0.7 | SA/HW | CumulativeMeans | 323.265 | 144.955 | 0 |
| QNet | 0.8 | SA/HW | Schruben | 508.222 | 145.219 | 0 |
| QNet | 0.8 | SA/HW | CumulativeMeans | 687.452 | 144.296 | 0 |
| QNet | 0.9 | SA/HW | Schruben | 1775.51 | 145.203 | 0 |
| QNet | 0.9 | SA/HW | CumulativeMeans | 2241.77 | 144.789 | 0 |
| QNet | 0.95 | SA/HW | Schruben | 6717.11 | 144.578 | 0 |
| QNet | 0.95 | SA/HW | CumulativeMeans | 7821.57 | 143.807 | 0 |
| AR(1) | 0.5 | MSE-DPBM | CumulativeMeans | 68.9344 | 15.022 | 110.056 |
| AR(1) | 0.5 | MSE-DPBM | Schruben | 68.9753 | 15.0224 | 110.131 |
| AR(1) | 0.6 | MSE-DPBM | CumulativeMeans | 68.2947 | 15.0117 | 109.038 |
| AR(1) | 0.6 | MSE-DPBM | Schruben | 68.2447 | 15.0128 | 108.978 |
| AR(1) | 0.7 | MSE-DPBM | Schruben | 66.9611 | 15.0082 | 106.959 |
| AR(1) | 0.7 | MSE-DPBM | CumulativeMeans | 66.9124 | 15.0071 | 106.896 |
| AR(1) | 0.8 | MSE-DPBM | Schruben | 63.693 | 15.004 | 101.838 |
| AR(1) | 0.8 | MSE-DPBM | CumulativeMeans | 63.9297 | 15.002 | 102.218 |
| AR(1) | 0.9 | MSE-DPBM | Schruben | 43.6855 | 15.0003 | 70.2802 |
| AR(1) | 0.9 | MSE-DPBM | CumulativeMeans | 44.8332 | 15.0002 | 72.0925 |
| AR(1) | 0.95 | MSE-DPBM | Schruben | 9.18715 | 15 | 15.6686 |
| AR(1) | 0.95 | MSE-DPBM | CumulativeMeans | 9.58601 | 15 | 16.302 |
| $M/H_2/1$ | 0.5 | MSE-DPBM | CumulativeMeans | 5226.38 | 16.1656 | 8440.96 |
| $M/H_2/1$ | 0.5 | MSE-DPBM | Schruben | 5224.79 | 16.1865 | 8440.73 |
| $M/H_2/1$ | 0.6 | MSE-DPBM | CumulativeMeans | 7859.32 | 16.1238 | 12693.2 |
| $M/H_2/1$ | 0.6 | MSE-DPBM | Schruben | 7877.7 | 16.1197 | 12719.8 |
| $M/H_2/1$ | 0.7 | MSE-DPBM | CumulativeMeans | 13026.6 | 16.1044 | 21025.7 |
| $M/H_2/1$ | 0.7 | MSE-DPBM | Schruben | 12946.3 | 16.1003 | 20899.2 |
| $M/H_2/1$ | 0.8 | MSE-DPBM | CumulativeMeans | 25982.6 | 16.156 | 41991.8 |
| $M/H_2/1$ | 0.8 | MSE-DPBM | Schruben | 25696.3 | 16.1564 | 41536.9 |

| | | | | | | |
|---|---|---|---|---|---|---|
| $M/H_2/1$ | 0.9 | MSE-DPBM | Schruben | 79000.5 | 16.139 | 127859 |
| $M/H_2/1$ | 0.9 | MSE-DPBM | CumulativeMeans | 82560.4 | 16.2119 | 133640 |
| $M/H_2/1$ | 0.95 | MSE-DPBM | Schruben | 244123 | 16.0368 | 395709 |
| $M/H_2/1$ | 0.95 | MSE-DPBM | CumulativeMeans | 252555 | 16.0765 | 409484 |
| M/D/1 | 0.5 | MSE-DPBM | Schruben | 85.7263 | 15.0109 | 136.591 |
| M/D/1 | 0.5 | MSE-DPBM | CumulativeMeans | 85.7823 | 15.0043 | 136.634 |
| M/D/1 | 0.6 | MSE-DPBM | Schruben | 200.131 | 15.0348 | 317.621 |
| M/D/1 | 0.6 | MSE-DPBM | CumulativeMeans | 201.172 | 15.0329 | 319.332 |
| M/D/1 | 0.7 | MSE-DPBM | CumulativeMeans | 514.745 | 15.1208 | 816.871 |
| M/D/1 | 0.7 | MSE-DPBM | Schruben | 514.093 | 15.1348 | 815.962 |
| M/D/1 | 0.8 | MSE-DPBM | CumulativeMeans | 1646.54 | 15.3202 | 2620.56 |
| M/D/1 | 0.8 | MSE-DPBM | Schruben | 1642.83 | 15.3422 | 2615.38 |
| M/D/1 | 0.9 | MSE-DPBM | CumulativeMeans | 8786.39 | 15.7422 | 14093.4 |
| M/D/1 | 0.9 | MSE-DPBM | Schruben | 8939.04 | 15.8303 | 14360.1 |
| M/D/1 | 0.95 | MSE-DPBM | CumulativeMeans | 35264.9 | 15.9532 | 56955.2 |
| M/D/1 | 0.95 | MSE-DPBM | Schruben | 39460.8 | 16.0547 | 63706.9 |
| M/M/1 | 0.5 | MSE-DPBM | Schruben | 599.217 | 15.3205 | 954.466 |
| M/M/1 | 0.5 | MSE-DPBM | CumulativeMeans | 601.663 | 15.3282 | 958.152 |
| M/M/1 | 0.6 | MSE-DPBM | CumulativeMeans | 1049.2 | 15.4591 | 1674.55 |
| M/M/1 | 0.6 | MSE-DPBM | Schruben | 1059.95 | 15.4475 | 1691.58 |
| M/M/1 | 0.7 | MSE-DPBM | Schruben | 2085.48 | 15.629 | 3337.78 |
| M/M/1 | 0.7 | MSE-DPBM | CumulativeMeans | 2097.75 | 15.6173 | 3356.94 |
| M/M/1 | 0.8 | MSE-DPBM | CumulativeMeans | 5199.61 | 15.8172 | 8350.47 |
| M/M/1 | 0.8 | MSE-DPBM | Schruben | 5198.41 | 15.8728 | 8352.72 |
| M/M/1 | 0.9 | MSE-DPBM | CumulativeMeans | 21208.2 | 15.9916 | 34192.5 |
| M/M/1 | 0.9 | MSE-DPBM | Schruben | 22494.6 | 16.067 | 36297.3 |
| M/M/1 | 0.95 | MSE-DPBM | CumulativeMeans | 70269.9 | 15.9672 | 113849 |
| M/M/1 | 0.95 | MSE-DPBM | Schruben | 83345.7 | 16.104 | 134871 |
| QNet | 0.5 | MSE-DPBM | CumulativeMeans | 440.692 | 15.1632 | 700.732 |
| QNet | 0.5 | MSE-DPBM | Schruben | 459.65 | 15.1936 | 731.014 |

| | | | | | | |
|---|---|---|---|---|---|---|
| QNet | 0.6 | MSE-DPBM | CumulativeMeans | 729.66 | 15.1796 | 1159.63 |
| QNet | 0.6 | MSE-DPBM | Schruben | 744.113 | 15.257 | 1183.7 |
| QNet | 0.7 | MSE-DPBM | Schruben | 1352.79 | 15.2942 | 2152.89 |
| QNet | 0.7 | MSE-DPBM | CumulativeMeans | 1336.02 | 15.2231 | 2123.85 |
| QNet | 0.8 | MSE-DPBM | Schruben | 3041.91 | 15.3363 | 4843.26 |
| QNet | 0.8 | MSE-DPBM | CumulativeMeans | 3056.53 | 15.2369 | 4859.35 |
| QNet | 0.9 | MSE-DPBM | CumulativeMeans | 11081 | 15.2308 | 17607 |
| QNet | 0.9 | MSE-DPBM | Schruben | 10910.8 | 15.3586 | 17372.7 |
| QNet | 0.95 | MSE-DPBM | Schruben | 41636 | 15.4084 | 66384.7 |
| QNet | 0.95 | MSE-DPBM | CumulativeMeans | 42130.4 | 15.2622 | 66949.4 |
| AR(1) | 0.5 | Mod. MSE-DPBM | Schruben | 253.227 | 15.0217 | 391.575 |
| AR(1) | 0.5 | Mod. MSE-DPBM | CumulativeMeans | 252.759 | 15.0253 | 393.528 |
| AR(1) | 0.6 | Mod. MSE-DPBM | Schruben | 252.673 | 15.026 | 391.039 |
| AR(1) | 0.6 | Mod. MSE-DPBM | CumulativeMeans | 252.691 | 15.0259 | 394.304 |
| AR(1) | 0.7 | Mod. MSE-DPBM | Schruben | 252.291 | 15.029 | 391.337 |
| AR(1) | 0.7 | Mod. MSE-DPBM | CumulativeMeans | 252.764 | 15.0253 | 394.061 |
| AR(1) | 0.8 | Mod. MSE-DPBM | Schruben | 252.788 | 15.0251 | 390.937 |
| AR(1) | 0.8 | Mod. MSE-DPBM | CumulativeMeans | 252.643 | 15.0262 | 392.524 |
| AR(1) | 0.9 | Mod. MSE-DPBM | Schruben | 251.543 | 15.0348 | 388.471 |
| AR(1) | 0.9 | Mod. MSE-DPBM | CumulativeMeans | 252.175 | 15.0299 | 388.809 |
| AR(1) | 0.95 | Mod. MSE-DPBM | Schruben | 250.282 | 15.0447 | 386.994 |
| AR(1) | 0.95 | Mod. MSE-DPBM | CumulativeMeans | 250.289 | 15.0446 | 385.812 |
| $M/H_2/1$ | 0.5 | Mod. MSE-DPBM | Schruben | 4236.53 | 29.9283 | 6650.87 |
| $M/H_2/1$ | 0.5 | Mod. MSE-DPBM | CumulativeMeans | 4173.45 | 30.2809 | 6556.81 |
| $M/H_2/1$ | 0.6 | Mod. MSE-DPBM | Schruben | 6225.53 | 30.2033 | 9858.64 |
| $M/H_2/1$ | 0.6 | Mod. MSE-DPBM | CumulativeMeans | 6036.69 | 30.9591 | 9616.42 |
| $M/H_2/1$ | 0.7 | Mod. MSE-DPBM | Schruben | 9702.76 | 33.1762 | 15446.5 |
| $M/H_2/1$ | 0.7 | Mod. MSE-DPBM | CumulativeMeans | 9508.02 | 33.6749 | 15183.5 |
| $M/H_2/1$ | 0.8 | Mod. MSE-DPBM | Schruben | 18539.6 | 37.1869 | 30088.6 |
| $M/H_2/1$ | 0.8 | Mod. MSE-DPBM | CumulativeMeans | 18295.6 | 37.0463 | 29673.1 |

| | | | | | | |
|---|---|---|---|---|---|---|
| $M/H_2/1$ | 0.9 | Mod. MSE-DPBM | Schruben | 53294 | 52.3898 | 90451.7 |
| $M/H_2/1$ | 0.9 | Mod. MSE-DPBM | CumulativeMeans | 54344.9 | 48.0373 | 92507 |
| $M/H_2/1$ | 0.95 | Mod. MSE-DPBM | Schruben | 155423 | 82.3599 | 273302 |
| $M/H_2/1$ | 0.95 | Mod. MSE-DPBM | CumulativeMeans | 158084 | 78.9475 | 275922 |
| M/D/1 | 0.5 | Mod. MSE-DPBM | Schruben | 245.611 | 15.0904 | 393.685 |
| M/D/1 | 0.5 | Mod. MSE-DPBM | CumulativeMeans | 245.573 | 15.0945 | 392.33 |
| M/D/1 | 0.6 | Mod. MSE-DPBM | Schruben | 257.906 | 15.3882 | 421.386 |
| M/D/1 | 0.6 | Mod. MSE-DPBM | CumulativeMeans | 257.947 | 15.3951 | 419.633 |
| M/D/1 | 0.7 | Mod. MSE-DPBM | Schruben | 469.86 | 17.1621 | 748.13 |
| M/D/1 | 0.7 | Mod. MSE-DPBM | CumulativeMeans | 463.395 | 17.5647 | 736.715 |
| M/D/1 | 0.8 | Mod. MSE-DPBM | Schruben | 1363.77 | 21.7975 | 2145.86 |
| M/D/1 | 0.8 | Mod. MSE-DPBM | CumulativeMeans | 1363.01 | 21.5574 | 2127.63 |
| M/D/1 | 0.9 | Mod. MSE-DPBM | Schruben | 6693.37 | 30.4543 | 10832.8 |
| M/D/1 | 0.9 | Mod. MSE-DPBM | CumulativeMeans | 6609.11 | 31.4588 | 10715.5 |
| M/D/1 | 0.95 | Mod. MSE-DPBM | Schruben | 27852 | 42.6348 | 46580.2 |
| M/D/1 | 0.95 | Mod. MSE-DPBM | CumulativeMeans | 27523.8 | 42.586 | 45954.4 |
| M/M/1 | 0.5 | Mod. MSE-DPBM | Schruben | 543.303 | 17.9296 | 841.269 |
| M/M/1 | 0.5 | Mod. MSE-DPBM | CumulativeMeans | 541.538 | 18.0376 | 833.442 |
| M/M/1 | 0.6 | Mod. MSE-DPBM | Schruben | 914.787 | 20.4575 | 1412.13 |
| M/M/1 | 0.6 | Mod. MSE-DPBM | CumulativeMeans | 908.424 | 20.3704 | 1417.33 |
| M/M/1 | 0.7 | Mod. MSE-DPBM | Schruben | 1714.02 | 24.0853 | 2674.49 |
| M/M/1 | 0.7 | Mod. MSE-DPBM | CumulativeMeans | 1717.45 | 24.1355 | 2678.25 |
| M/M/1 | 0.8 | Mod. MSE-DPBM | Schruben | 4111.96 | 28.2705 | 6535.01 |
| M/M/1 | 0.8 | Mod. MSE-DPBM | CumulativeMeans | 4081.07 | 27.8652 | 6471.03 |
| M/M/1 | 0.9 | Mod. MSE-DPBM | Schruben | 15976.9 | 36.6498 | 26279.7 |
| M/M/1 | 0.9 | Mod. MSE-DPBM | CumulativeMeans | 15907.7 | 37.0128 | 26106.1 |
| M/M/1 | 0.95 | Mod. MSE-DPBM | Schruben | 54790.7 | 57.9878 | 92587.8 |
| M/M/1 | 0.95 | Mod. MSE-DPBM | CumulativeMeans | 55414.3 | 55.0286 | 92879.8 |
| QNet | 0.5 | Mod. MSE-DPBM | Schruben | 428.068 | 16.7505 | 632.463 |
| QNet | 0.5 | Mod. MSE-DPBM | CumulativeMeans | 396.948 | 17.285 | 550.754 |

| | | | | | | |
|---|---|---|---|---|---|---|
| QNet | 0.6 | Mod. MSE-DPBM | Schruben | 656.925 | 18.8729 | 961.55 |
| QNet | 0.6 | Mod. MSE-DPBM | CumulativeMeans | 614.186 | 19.7247 | 846.012 |
| QNet | 0.7 | Mod. MSE-DPBM | Schruben | 1113.41 | 22.3059 | 1618.1 |
| QNet | 0.7 | Mod. MSE-DPBM | CumulativeMeans | 1047.29 | 23.6548 | 1478.59 |
| QNet | 0.8 | Mod. MSE-DPBM | Schruben | 2238.68 | 28.3639 | 3319.08 |
| QNet | 0.8 | Mod. MSE-DPBM | CumulativeMeans | 2165.23 | 29.8233 | 3156.1 |
| QNet | 0.9 | Mod. MSE-DPBM | Schruben | 6564.88 | 42.3196 | 9995.59 |
| QNet | 0.9 | Mod. MSE-DPBM | CumulativeMeans | 6415.36 | 43.4503 | 9927.61 |
| QNet | 0.95 | Mod. MSE-DPBM | Schruben | 20286.4 | 59.2605 | 32638.3 |
| QNet | 0.95 | Mod. MSE-DPBM | CumulativeMeans | 19427.1 | 60.7532 | 31389.8 |

Table E.6: Avearge batch size and number of batches per method

# Appendix F

# Akaroa2 Method Registration

In order to use the methods of simulation output analysis they have to register themselves as available methods under Akaroa2 and be compiled with the whole application. The code in Appendix A, Appendix B and Appendix C are necessary. Their object file path have to be added to AKANAL_OBJ in Makefile.main.