**MASTER IN COMPUTER SCIENCE**

# Development of a quasi-birth-and-death sensitivity analysis tool

Cordy, Maxime

*Award date:*
2011

*Awarding institution:*
University of Namur

Link to publication

Facultés Universitaires Notre-Dame de la Paix, Namur

Faculté d'informatique

Année académique 2010-2011

# Development of a Quasi-Birth-and-Death Sensitivity Analysis Tool

Maxime Cordy

Mémoire présenté en vue de l'obtention du grade de
Master en Sciences Informatiques

## Abstract

In the past, the performance analysis of random systems has been of great interest. Such an analysis often consists in the computation of so-called performance measures. Sometimes, it is desired to evaluate the sensitivity of the performance with respect to variation applied to the parameters of the considered system. Such an evaluation is usually referred to as sensitivity analysis.

Markov processes are intensively used to model the evolution of random systems over time. We are especially interested in a particular case of Markov processes, called the Quasi-Birth-and-Death process (QBD). The QBDs combine a nice modelling expressiveness with the possibility of using efficient evaluation methods.

We aim the development of a tool that can perform sensitivity analyses on QBDs. More precisely, the tool is required to provide methods to specify a QBD process, procedures to carry out the analyses, and interfaces to display the produced results. To achieve this, we present the theoretical foundations of methods to compute the performance measures, as well as the tool and a case study made with it.

**Keywords**: performance evaluation, sensitivity analysis, Markov process, Quasi-Birth-and-Death, analysis tool

## Résumé

Par le passé, l'analyse de performance de systèmes aléatoires a été d'un grand intérêt. Mener une telle analyse consiste souvent à calculer des mesures de performance. Parfois, on peut vouloir évaluer la sensibilité de la performance par rapport à des variations des paramètres du système considéré. Une telle évaluation est généralement nommée "analyse de sensibilité".

Les processus de Markov sont souvent utilisés pour modéliser l'évolution de systèmes aléatoires. Nous nous intéressons à un cas particulier de ces processus, à savoir le processus de Quasi-Birth-and-Death (QBD). Les QBD combine une bonne expressivité de modélisation avec la possibilité d'utiliser des méthodes d'évaluation efficaces.

Notre objectif est le développement d'un outil qui peut analyser la sensibilité des QBD. Plus précisément, l'outil doit fournir des méthodes pour spécifier un tel processus, des procédures pour mener les analyses et des interfaces pour afficher les résultats produits. A cette fin, nous présentons les fondements théoriques de méthodes pour calculer des mesures de performance, ainsi que l'outil et une étude de cas réalisée avec celui-ci.

**Mots-clés**: évaluation de performance, analyse de sensibilité, processus de Markov, Quasi-Birth-and-Death, outil d'analyse

# Acknowledgments

First of all, I would like to acknowledge the people who have supported me in a way or another through the writing of this master's thesis.

In particular, I sincerely thank Prof. Marie-Ange Remiche, my supervisor, for her precious advice and the interesting discussions we had. Her comments about the writing significantly contributed to the improvement of the textual quality of the master's thesis. I also thank her for giving me the opportunity to present my work at two conferences. These experiences are invaluable to me.

I absolutely want to acknowledge Dr. Julian Dronckier for the time he spent reading the whole text, correcting it and improving my english writing skills. Thanks to his help, I was able to increase the quality of the master's thesis.

Also, I would like to thank Teh Amouh and Maher Chemseddine for their warm welcome and all the nice discussions we had during my four month research work at the University of Namur.

Finally, I thank my close relatives and friends for all their support during the writing of this master's thesis.

# Contents

**Introduction** **xv**

**I Fundamental Definitions** **1**

**1 Quasi-Birth-and-Death** **3**

1.1 Random variable . . . . . . . . . . 4

1.1.1 Discrete random variable . . . . . . . . . . 4

1.1.2 Continuous random variable . . . . . . . . . . 6

1.2 Markov process . . . . . . . . . . 10

1.2.1 Discrete time Markov process . . . . . . . . . . 11

1.2.2 Continuous time Markov process . . . . . . . . . . 13

1.3 Quasi-Birth-and-Death process . . . . . . . . . . 16

1.3.1 Definition & Generator . . . . . . . . . . 17

1.3.2 The M/PH/1 queue . . . . . . . . . . 19

1.3.3 Uniformization of Markov processes . . . . . . . . . . 22

1.3.4 Kendall's notation . . . . . . . . . . 25

1.4 Conclusion . . . . . . . . . . 26

**II Computation Methods** **27**

**2 Matrix Analytic Methods** **29**

2.1 Stationary Distribution . . . . . . . . . . 29

2.1.1 Stability conditions . . . . . . . . . . 30

2.1.2 Gaussian Elimination algorithm . . . . . . . . . . 33

2.1.3 Logarithmic Reduction algorithm . . . . . . . . . . 34

2.1.4 Linear Level Reduction algorithm . . . . . . . . . . 42

2.2 First passage times . . . . . . . . . . 46

2.2.1 Computing first passage times upward . . . . . . . . . . 48

2.2.2 Computing first passage times downward . . . . . . . . . . 50

2.3 Conclusion . . . . . . . . . . 51

**3 Discrete-Event Simulation** **53**
3.1 Next-event simulation . . . . . . . . . . . . . . . . . . . . . . 54
3.2 Simulating a Quasi-Birth-Death process . . . . . . . . . . . . 56
3.3 Random number generator . . . . . . . . . . . . . . . . . . . . 59
3.3.1 Linear congruential generator . . . . . . . . . . . . . . . 59
3.3.2 Generating discrete random variables . . . . . . . . . . 63
3.3.3 Generating continuous random variables . . . . . . . . 64
3.4 Output analysis and data estimation . . . . . . . . . . . . . . 66
3.4.1 Estimation of the performance measures . . . . . . . . 66
3.4.2 Steady-state simulation . . . . . . . . . . . . . . . . . . 67
3.4.3 Confidence intervals . . . . . . . . . . . . . . . . . . . . 69
3.5 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 72

**III Tool implementation** **73**

**4 QBDSense Tool** **75**
4.1 General architecture and functionalities . . . . . . . . . . . . 76
4.2 Description of the input interfaces . . . . . . . . . . . . . . . 77
4.2.1 Method 1: Explicit definition of the inner blocks . . . 79
4.2.2 Method 2: Definition of a queueing system . . . . . . . 79
4.2.3 Method 3: Specification of the transitions . . . . . . . 80
4.2.4 Parameterisation of a sensitivity analysis . . . . . . . . 89
4.3 The performance analysis frameworks . . . . . . . . . . . . . 91
4.3.1 QBDSolver . . . . . . . . . . . . . . . . . . . . . . . . . 91
4.3.2 QBDSimulator . . . . . . . . . . . . . . . . . . . . . . . 92
4.4 Outputs display . . . . . . . . . . . . . . . . . . . . . . . . . . 94
4.5 Architecture of the tool . . . . . . . . . . . . . . . . . . . . . . 96
4.6 Limits and perspectives . . . . . . . . . . . . . . . . . . . . . . 98

**5 Case Study** **101**
5.1 System description . . . . . . . . . . . . . . . . . . . . . . . . . 101
5.2 Mathematical model . . . . . . . . . . . . . . . . . . . . . . . . 103
5.3 An approach to model a composite system . . . . . . . . . . . 104
5.3.1 Kronecker product and sum . . . . . . . . . . . . . . . . 105
5.3.2 Definition of a simple repairable system . . . . . . . . 105
5.3.3 Modelling the simple repairable system . . . . . . . . . 106
5.4 Specification of the infinitesimal generator . . . . . . . . . . . 108
5.4.1 Transitions from level 0 to level 1 . . . . . . . . . . . . 108
5.4.2 Transitions from level 0 to level 0 . . . . . . . . . . . . 112
5.5 Numerical evaluation . . . . . . . . . . . . . . . . . . . . . . . . 114

**IV Conclusion 119**

**6 Review and perspectives 121**
6.1 Work Summary . . . . . 121
6.2 Contributions of the thesis . . . . . 122
6.3 Limitations of our approaches . . . . . 123
6.4 Perspectives and challenges . . . . . 123

**V Appendix 125**

**A ORBEL 25 Paper 127**

**B ValueTools 2011 Paper 131**

**C Case study: original article 141**

# List of Definitions

1.1.1 Discrete probability function . . . . . . . . . . . . . 4

1.1.2 Probability of an event . . . . . . . . . . . . . . . . . . 4

1.1.3 Discrete random variable . . . . . . . . . . . . . . . . . 5

1.1.4 Probability mass function . . . . . . . . . . . . . . . . 6

1.1.5 Continuous random variable . . . . . . . . . . . . . . . 7

1.1.6 Cumulative distribution function . . . . . . . . . . . 7

1.1.7 Exponential distribution . . . . . . . . . . . . . . . . . 8

1.1.8 Normal distribution . . . . . . . . . . . . . . . . . . . 8

1.1.9 Student's t-distribution . . . . . . . . . . . . . . . . . 9

1.2.1 Stochastic process . . . . . . . . . . . . . . . . . . . . 10

1.2.2 Conditional probability . . . . . . . . . . . . . . . . . 11

1.2.3 Discrete time Markov process . . . . . . . . . . . . . 11

1.2.4 Continuous time Markov process . . . . . . . . . . . 13

1.2.5 Phase-type distribution . . . . . . . . . . . . . . . . . 15

1.3.1 Kendall's notation . . . . . . . . . . . . . . . . . . . . 26

2.1.1 Period . . . . . . . . . . . . . . . . . . . . . . . . . . . 30

2.1.2 Recurrent state . . . . . . . . . . . . . . . . . . . . . . 31

2.1.3 Positive & Null recurrent state . . . . . . . . . . . . 31

2.1.4 Accessible state . . . . . . . . . . . . . . . . . . . . . 31

2.1.5 Communicating state . . . . . . . . . . . . . . . . . . 32

2.1.6 Equivalence class . . . . . . . . . . . . . . . . . . . . 32

2.1.7 Spectral radius . . . . . . . . . . . . . . . . . . . . . . 37

2.2.1 Expected first passage time . . . . . . . . . . . . . . 47

3.1.1 Discrete-event simulation model . . . . . . . . . . . . 54

3.3.1 Lehmer's algorithm . . . . . . . . . . . . . . . . . . . 60

3.3.2 Fundamental period . . . . . . . . . . . . . . . . . . . 61

3.3.3 Full-period generator . . . . . . . . . . . . . . . . . . . 62
3.3.4 Inverse distribution function (discrete) . . . . . . . . 63
3.3.5 Inverse distribution function (continuous) . . . . . . 65
4.2.1 Similar Macrostates Group . . . . . . . . . . . . . . . 84
4.2.2 Fusion of two Similar Macrostates Groups . . . . . . 86
5.3.1 Kronecker product . . . . . . . . . . . . . . . . . . . . 105
5.3.2 Kronecker sum . . . . . . . . . . . . . . . . . . . . . . 105

# List of Theorems

2.1.1 Homogeneity in an equivalence class . . . . . . . . . 32

2.1.2 Stability conditions (discrete time) . . . . . . . . . . 33

2.1.3 Stability conditions (continuous time) . . . . . . . . 33

2.1.4 Rate matrix existence . . . . . . . . . . . . . . . . . . 36

2.1.5 Convergence of the series $\sum_{k\geq 0} M^k$ . . . . . . . . . . 38

2.1.6 Relations between the matrices $R$, $G$, and $U$ . . . . . 40

2.1.7 Computation of matrix $G$ . . . . . . . . . . . . . . . 41

2.1.8 Determination of $\pi$ (finite and homogeneous) . . . . 43

2.1.9 Determination of $\pi$ (inhomogeneous) . . . . . . . . . 45

2.2.1 Computation of upward first passage times . . . . . 49

3.3.1 Full-period conditions ($b \neq 0$) . . . . . . . . . . . . . 62

3.3.2 Full-period conditions ($b = 0$) . . . . . . . . . . . . . 63

3.3.3 Probability integral transformation (discrete) . . . . 64

3.3.4 Probability integral transformation (continuous) . . . . . 65

3.4.1 Central-limit theorem . . . . . . . . . . . . . . . . . . 68

3.4.2 Student's test . . . . . . . . . . . . . . . . . . . . . . . 71

# List of Figures

1.1 The cumulative distribution function of two exponential distributions, one with rate 2 (blue curve) and one rate 1 (green curve). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
1.2 The possible transitions of the stochastic process presented in Example 1.2.1. . . . . . . . . . . . . . . . . . . . . . . . . . . 13
1.3 A continuous time Markov process with an absorbing state. . 15
1.4 The available transitions in a Quasi-Birth-and-Death process. 18
1.5 The possible transitions of a QBD modelling an M/PH/1 queueing system. . . . . . . . . . . . . . . . . . . . . . . . . 21

2.1 An OCTAVE function that solves the system $\pi = \pi P, \pi\mathbf{1} = 1$, based on Gaussian elimination. . . . . . . . . . . . . . . . . . 35
2.2 An OCTAVE function that implements the *Logarithmic Reduction* algorithm. . . . . . . . . . . . . . . . . . . . . . . . . 42
2.3 An OCTAVE function that computes the stationary vector of a homogeneous and finite QBD. . . . . . . . . . . . . . . . . . 44
2.4 An OCTAVE function that computes the stationary vector of an inhomogeneous and finite QBD. . . . . . . . . . . . . . . . 46
2.5 An OCTAVE function that computes the first passage time from level $S$ to level $M$ of an inhomogeneous QBD. . . . . . . 50

3.1 An OCTAVE function that simulates a Markov process. . . . . 57
3.2 An OCTAVE function to simulate a Quasi-Birth-and-Death process. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 59
3.3 An OCTAVE function that builds a confidence interval. . . . . 71

4.1 The possible transitions in the M/PH/1 queueing system of Example 4.2.1. . . . . . . . . . . . . . . . . . . . . . . . . . . 78
4.2 An Octave function that computes the inner block composing the generator of a QBD modeling an M/Ph/1 queue. . . . . . 80
4.3 A graphical user interface used to specify a QBD as a queueing system. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 81
4.4 Specification of a QBD with a textual language. . . . . . . . . 84

4.5 An OCTAVE function that computes the inner blocks composing the generator of a QBD modelling a **perturbed** M/Ph/1 queue. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 90
4.6 A graphical user interface used to define a perturbation on a queueing system. . . . . . . . . . . . . . . . . . . . . . . . . . 90
4.7 The stationary probabilities of the the first six levels of the QBD defined in Example 4.2.1 . . . . . . . . . . . . . . . . . . 95
4.8 The stationary probabilities of the first six levels of Example 4.2.1 estimated by independent replications of a simulation. . 95
4.9 The architecture of the tool illustrated with a UML component diagram. . . . . . . . . . . . . . . . . . . . . . . . . . . . 97

5.1 A repairable system under degradation, inspection, and two types of repair. . . . . . . . . . . . . . . . . . . . . . . . . . . 102
5.2 Evolution of the ROCOF of the repairable system (chart). . . 118

# Introduction

Performance analysis can be defined as a quantitative study evaluating how a given system succeeds in reaching the achievements for which it was built. For example, the performance of a web server can be defined as the number of requests it can fulfill per second; the performance of an internet service provider can be determined as its mean availability percentage; and the performance of a production line can refer to the mean creation time of a single product. Sometimes, it is also aimed to assess the sensitivity of the performance with respect to variations applied to the parameters of the considered system. Such an evaluation is usually referred to as sensitivity analysis.

During the last decades, performance and sensitivity analyses have been of great interest. It has a huge importance not only for university researchers but also for industrials. For example, IBM involved a performance analysis in the development of the Time Sharing Option (TSO) for their operating system IBM 360 [26]. Nowadays, performance is still at the heart of concerns, in particular in finance, in network systems, and in intelligent systems. Particularly in computer science, the performance of systems has been theoretically studied since the early 1960 up until now. A large variety of systems has been intensively studied, notably time-shared systems [15], packet-switching networks [16], and more recently, Internet-based Television [13] and peer-to-peer systems [11].

These types of systems have common features. First, time has a significant impact on them. Second, their state is influenced by the outcome of so-called random events and their respective probability law. Subsequently, the analysis of the performance of such systems is hardened. Leemis and Park [19] propose the following steps toward the determination of the performance of random systems:

1. To determine the goals and the objectives of the analysis, notably, which answers the analysis is supposed to provide.

2. To build a conceptual model of the system that is based on the defined goals. The conceptual model is an abstraction of the real system. In

particular, this step consists in the determination of what is relevant to observe and what is negligible enough to be ignored.

3. To turn the conceptual model into a specification model. Basically, either the random events impacting on the system are represented by statistical data measured on the real system, or these events are modelled using a so-called stochastic model.

4. To develop a computational model with respect to the specification model. The computational model essentially consists in a computer program that returns the expected performance values.

5. To verify the computational model. The computer program should be consistent with regard to the specification model.

6. To validate the computational model. Although it is verified, a computational model can still be inconsistent with the real system that is studied. This step must thus confirm that we built the right model.

Although each of these procedures is essential for a relevant performance analysis, this master's thesis only deals with steps 3, 4 and 5.

More precisely, we focus on the building of specification models based on stochastic models. For this purpose, we present the fundamental concepts and definitions of stochastic modelling. In particular, the Markov process is a well-known type of stochastic model that is intensively used to model the evolution of random systems over time. A specific feature of the Markov process is the so-called Markov property. When this property holds, the next state of the process can be determined knowing only its current state but not its complete history. Furthermore, when the process has a given set of properties, we can compute theoretical measures often referred to as steady state measures. We focus on this type of measures because we can derive from them performance measures relevant for the studied system.

In the thesis, we are especially interested in a particular case of Markov process, called the Quasi-Birth-and-Death process (QBD). QBDs are Markov processes whose state is defined by two variables. They are such that restrictions are applied to their possible transitions. In spite of these restrictions, they have a good modelling expressiveness. Indeed, they have been extensively used in the past to evaluate the performance of a wide variety of systems, in particular reliability systems [24] and peer-to-peer systems [11].

The final objective of the thesis is the development of a tool that permits to analyse the performance of a system modelled as a Quasi-Birth-and-Death process. More precisely, the tool must include the following features:

- formal methods to specify a Quasi-Birth-and-Death process,
- the possibility to analyse the performance of the process and to study its sensitivity with regard to small variations applied to its parameters,
- two methods to compute the measures, *i.e.* explicit computations and estimations via simulations.

The development of the specification methods is essential. These methods must allow to concisely and easily define complex QBDs. In particular, one of the method we present is based on the principles of the language theory. More precisely, we define a textual language to specify the transitions of a QBD and we build a parser that defines the process to analyse from the transitions specification.

It is noteworthy that we deal not only with the evaluation of a system, but also with the analysis of its sensitivity. More precisely, instead of merely computing performance measures related to a given stochastic model of the system, the tool allows to observe how the value of these measures evolves when some parameters of the model are slightly modified.

Another important part of the thesis is the study of algorithms to compute the value of steady state measures related to the specification model. While the computation of these measures is generally time and space consuming, it is considerably eased when the considered Markov process is a QBD. Thanks to the particular structure of such a process, efficient algorithms have been developed to explicitly compute the steady state measures. They are based on the so-called matrix analytic methods [18]. Because there are different types of QBDs, there exist several algorithms to compute a given measure. Furthermore, optimized versions of these algorithms that can be applied only when the QBD has specific properties can be found in the scientific literature. We therefore limit our presentation to the most general algorithms. We also give the theoretical foundations of these algorithms and we provide references to the proof of their correctness.

Discrete-event simulation constitutes an alternative to the matrix analytic methods. Instead of explicitly computing the measures, the behaviour of the system is simulated and the steady state values are estimated from the simulation results. In particular, we present the principles of a discrete-event simulation approach called next-event and according to this approach, we implement an algorithm that simulates a Quasi-Birth-and-Death process. Unlike an explicit computation, a simulation does not require any costly matrix manipulations. However, estimation techniques must be used to obtain accurate data.

Both the matrix analytic methods and the simulation approaches lead us to the development of computational models, namely computer programs that provide an evaluation of the performance of a specification model such as they were previously defined. Furthermore, because the algorithms implemented in these computer programs are strongly coupled with proven mathematical theories, the verification step we mentioned earlier is considerably eased.

In summary, the purposes of the thesis are (i) a synthesis of both the matrix analytic methods and the discrete-event simulation approach, specifically applied to the Quasi-Birth-and-Death process, (ii) the definition of formal methods to specify a QBD, and (iii) the development of our tool, which makes use of the specification interfaces, as well as the two computation methods in order to perform sensitivity analyses on QBDs.

We structure the master's thesis as follows:

**Part I** gives the essential concepts and definitions related to stochastic modelling. Eventually, we define the type of specification model we aim to build, namely the Quasi-Birth-and-Death process. We also present definitions and techniques that are required for a thorough understanding of the thesis. This part includes only Chapter 1.

**Part II** is dedicated to the presentation of the computation algorithms. More precisely, Chapter 2 defines the statistical measures we are interested in and it presents the theoretical foundations of the matrix analytic methods applied to QBDs, which eventually lead to the development of algorithms to compute all the defined measures for each type of QBD. Next, Chapter 3 presents the principles of the discrete-event simulation approach, as well as the development of a QBD simulating algorithm we developed thanks to these principles and the formal definitions given in Chapter 1.

**Part III** finally presents our tool. In Chapter 4, we give the architecture of the tool. We especially put the emphasis on the input interfaces, *i.e.* the methods we develop to specify a QBD process. In particular, we present the syntax and the semantic of our language. We also describe how the algorithms presented in Chapter 2 and in Chapter 3 are integrated within our tool. Finally, we carry out a case study in Chapter 5. We start from a computational model found in the scientific literature, we build a specification model (*i.e.* a QBD process) accordingly, and we use our tool to perform a sensitivity analysis on a performance measure.

**Part IV** reviews our work. In particular, we put emphasis on our personal contributions and we give insight into future developments that would be of interest.

**Part V** is the appendix. It contains two articles related to the thesis, which were published in some conference proceedings. It also includes the original article on which our case study is based.

## Part I

# Fundamental Definitions

# Chapter 1

# Quasi-Birth-and-Death

In order to evaluate a system, it must be first modelled with an adequate mathematical model. This model represents both the system to analyse and the random events that influence it. Therefore, the model must fit the "occurrence tendency" of these events and respect the actual features of the system. For example, queue models are commonly used to model systems that randomly receive *requests*, or *jobs*, and complete them in a nondeterministic time. Nowadays, queue models are intensively used, especially in computer science. The state of a queueing system is often defined as the number of *jobs* in the queue or in service. The evolution of this number over time can be modelled as a type of random process, that is the *Markov process*. In this thesis, we mainly focus on a particular case of Markov process, called the *Quasi-Birth-and-Death* (QBD). As we will see, QBDs have a particular structure. Their features considerably help the evaluation of their performance, as they permit the use of specific algorithms we present in Chapters 2 and 3.

This chapter aims to define Quasi-Birth-and-Death processes and to give examples of systems whose state evolution can be modelled as a QBD. The chapter is presented as follows. We briefly introduce in Section 1.1 the basic definitions and the general ideas of the probability theory. We stress that our objective is not to present the probability theory thoroughly, but only to give an insight into concepts that are required for an understandable reading of the thesis. A well-known type of random process, the *Markov process*, is presented in Section 1.2. In Section 1.3, we put emphasis on the *Quasi-Birth-and-Death* process. Then, we specifically describe its particular structure. Finally, we illustrate its use by modelling a specific queueing system.

## 1.1 Random variable

We begin this section by a brief introduction to the probability theory. This branch of the mathematics is focused on the study of random phenomena that can have several outcomes. When such a phenomenon occurs, only one of its possible results is realized. The set of all the outcomes is called **the sample space**, usually noted $\Omega$. Any subset of the sample space is called an **event**. An intrinsic value is commonly given to every result of a phenomenon to express its likelihood. The higher the value, the more chances this result has to occur. This value is called **probability**. The assignation of a probability to the outcomes of a sample space is named **probability distribution**, often abbreviated as "distribution". Depending on whether or not the considered sample space is a countable set, a probability distribution is defined differently. First, we give a formal definition of probability distribution for a countable sample space. Then, we see why this first definition cannot hold for an uncountable sample space and subsequently, we give a new definition that is more convenient.

### 1.1.1 Discrete random variable

A discrete probability distribution gives a probability to every outcome of a countable sample space. In other words, a value is associated with each of the outcomes, describing its occurrence chances. We define this kind of probability distribution more formally and we illustrate it with an example.

**Definition 1.1.1 (Discrete probability function)**
*A* ***discrete probability function*** *defined on a countable sample space $\Omega$ is a function $f : \Omega \longrightarrow \mathbb{R}$ such that:*

- $\forall x \in \Omega : f(x) \in [0, 1]$
- $\displaystyle\sum_{x\in\Omega} f(x) = 1$

**Definition 1.1.2 (Probability of an event)**
*Let $\Omega$ be a countable sample space, $\mathcal{P}(\Omega)$ its powerset, and $f$, a probability function defined on it. The* ***probability of an event*** *$E$, i.e. a subset of $\Omega$, is a function $P : \mathcal{P}(\Omega) \longrightarrow R$ such that:*

- $P(E) = \displaystyle\sum_{\omega\in E} f(\omega)$

**Example 1.1.1** *RoyalJelly is a candy store that sells only bags of handmade candies. All the candies have an identical shape. However, during the fabrication process, each candy randomly receives a colour, which determines its flavour. The possible colours are: red, orange, yellow or green. Every colour has equal probability of appearing. Therefore, for the random phenomenon "colour of the candy", we have that*

$$\Omega = \{red, orange, yellow, green\},$$
$$P(red) = P(orange) = P(yellow) = P(green) = 0.25.$$

This example is quite basic. However, it will be used again to introduce new concepts in this chapter. Independently of their probability, the outcomes of a random phenomenon are often associated with another value. This value allows to define an order relation on the set of the outcomes. It can be used to express, for example, that an outcome is preferred over another one. It can also describe the state of a system affected by random phenomena, as we will see later. The function that defines such a value for each possible outcome is called **random variable**. There exist two types of random variables: the discrete ones and the continuous ones. For now, we focus only on discrete random variables. The continuous ones are discussed later in this chapter. More formally, we define a discrete random variable as follows.

**Definition 1.1.3 (Discrete random variable)**
*Let $\Omega$ be a sample space. A **discrete random variable** is a function $X : \Omega \longrightarrow \mathcal{S} \subset \mathbb{R}$, where $\mathcal{S}$ is a **countable** set. The set of the possible values returned by $X$, that is $\mathcal{S}$, is called the **state space** of $X$.*

Our definition of random variable is not the most general one. Indeed, there exist random variables that are defined on a state space other than a subset of $\mathbb{R}$. However, as those ones are out of the scope of this thesis, we do not consider them. It is also noteworthy that there can be no correlation between the probability of an outcome and the corresponding value of a random variable defined on its sample space.

If a random variable is defined on a countable sample space, its state space is necessarily countable. Hence, this random variable is discrete. When studying discrete random variables, we are often interested in the probability that they have a given value, hence in their probability distribution. When the random variable is discrete, its distribution can be defined by a **probability mass function**.

**Definition 1.1.4 (Probability mass function)**
*Let $\mathcal{S} = \{v_i\}$ be the state space of a discrete random variable $X$ defined on a sample space $\Omega$. Then, we call the* ***probability mass function*** *of $X$ the function $P_X : \mathcal{S} \longrightarrow [0, 1]$ such that:*

- $P_X(v_i) = P(X = v_i) \overset{\text{def}}{=} P(\{\omega \in \Omega : X(\omega) = v_i\})$
- $\sum_{v_i \in \mathcal{S}} P_X(v_i) = 1$

We illustrate these definitions with an extension of Example 1.1.1.

**Example 1.1.2** *RoyalJelly's shopkeeper grouped the candies by colour in the store. When a customer enters the shop, he gets an empty bag and chooses the colour of the candies to be put in the bag. Furthermore, the shopkeeper knows which flavours are generally preferred by the customers and decides to assign the price of every coloured candy according to its popularity:*

$$\begin{aligned} Price(red) &= 3, \\ Price(green) &= 2, \\ Price(orange) &= 2, \\ Price(yellow) &= 1. \end{aligned}$$

*Obviously, Price is a discrete random variable with state space $\{1, 2, 3\}$, as it assigns the value 1, 2 or 3 to every outcome of the random phenomenon "colour of the candy". As previously stated (see Example 1.1.1), every color has an equal probability to appear. Hence, the probability mass function associated with the random variable is defined as*

$$\begin{aligned} P_{Price}(3) &= 0.25, \\ P_{Price}(2) &= 0.5, \\ P_{Price}(1) &= 0.25. \end{aligned}$$

*Thus, a candy costs 3 money units if it is red, 2 units if it is green or orange, and 1 unit if it is yellow.*

## 1.1.2 Continuous random variable

We now give a definition of continuous random variables. Again, this definition concerns only a subset of all the random variables. However, we do not care about the other ones.

**Definition 1.1.5 (Continuous random variable)**
*Let $\Omega$ be an uncountable sample space. A* ***continuous random variable*** *is a function $X : \Omega \longrightarrow \mathcal{S} \subseteq \mathbb{R}$ where $\mathcal{S}$, called the* ***state space*** *of $X$, is a* ***uncountable*** *set.*

As stated before, the definition of discrete probability distribution cannot hold when it comes to an uncountable sample space. In this case, any single outcome of the random phenomenon has an infinitesimal probability, which statistically results in a zero probability. The same property holds when we consider the uncountable state space of continuous random variables. Therefore, there is an infinitesimal probability that these variables have a given value. From a statistical point of view, this probability is equal to zero. In particular, such random variables are often used to model measures like height, weight, and time. Because the state space is uncountable, the distribution of continuous random variables cannot be defined by a probability mass function. Instead, we define it with a **cumulative distribution function**.

**Definition 1.1.6 (Cumulative distribution function)**
*Let $\mathcal{S}$ be the state space of a continuous random variable $X$ defined on a sample space $\Omega$. The* ***cumulative distribution function (CDF)*** *of $X$ is the strictly ascending function $F_X : \mathcal{S} \longrightarrow [0, 1]$ such that:*

- $F_X(a) = P(X \leq a) \overset{\text{def}}{=} P(\{\omega \in \Omega | X(\omega) \leq a\})$
- $\lim\limits_{a \to -\infty} F_X(a) = 0$
- $\lim\limits_{a \to +\infty} F_X(a) = 1$

Cumulative distribution functions allow to characterize the probability distribution of continuous random variables. Basically, they can be used to compute the probability that the value of the variable is in a given interval. A continuous probability distribution can also be defined by the derivative of the cumulative distribution function, called **the probability density function**. We give examples of continuous distribution by presenting some of the most well-known. We will make use of these ones in the next chapters.

**Definition 1.1.7 (Exponential distribution)**
*The **exponential distribution** is a continuous probability distribution that is characterized by the following cumulative distribution function:*

$$F(x) = \begin{cases} 1 - e^{-\lambda x}, & x \geq 0, \\ 0, & x < 0. \end{cases} \tag{1.1}$$

*where $\lambda \in \mathbb{R}_0^+$ is called the **rate** of the exponential distribution.*

Exponential distributions have been extensively used to model events that occur at random time. In order to illustrate the impact of the rate value on the distribution, we give the following example.

**Example 1.1.3** *Figure 1.1 shows the CDF of two exponential distributions. The one drawn in blue has a rate of 2. The probability that the value of the random variable, e.g the time of occurrence, is between 0 and 0.5 is about 0.63. The occurrence time is in [0,1] with a probability of about 0.86, and it is in [0, 1.5] with a probability of about 0.95. The green curve represents another exponential distribution, this time with rate 1. It is noteworthy that the higher the rate, the smaller the expected occurrence time. Indeed, for any x, we see that the probability of the occurrence time being in [0, x] is higher on the blue curve than on the green one. Also, because a CDF is a strictly ascending function, the probability that an event occurs increases with the length of the considered interval [0, x]. However, this increase becomes slower and slower.*

There are two other continuous probability distributions that we require further in the thesis, in particular in Chapter 3. We give them a formal definition according to Leemis and Park [19, Chapter 7].

**Definition 1.1.8 (Normal distribution)**
*A **Normal** $(\mu, \sigma)$ distribution, with $\mu \in \mathbb{R}, \sigma \in \mathbb{R}_0^+$ is a continuous probability distribution defined by the following cumulative distribution function*

$$F(x) = \Phi(\frac{x - \mu}{\sigma}) \tag{1.2}$$

*with*

$$\Phi(z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} e^{-t^2/2} dt. \tag{1.3}$$

*Furthermore, $\mu$ is the mean of the distribution and $\sigma$ is its standard deviation.*

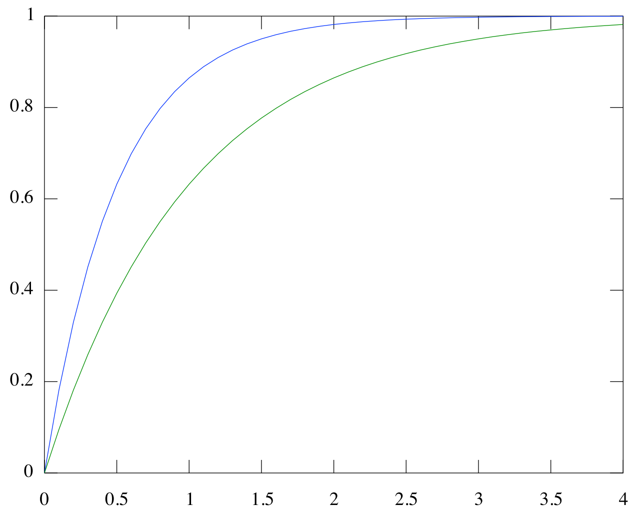


Figure 1.1: The cumulative distribution function of two exponential distributions, one with rate 2 (blue curve) and one rate 1 (green curve).

It is noteworthy that if a continuous random variable $X$ is Normal$(\mu, \sigma)$ distributed, then the random variable $(X - \mu)/\sigma$ is Normal$(0, 1)$ distributed.

Finally, we define the last continuous probability distribution of this section.

**Definition 1.1.9 (Student's t-distribution)**
*A* ***Student's t-distribution****, with parameter $n \in \mathbb{N}_0$, is a continuous probability distribution defined by the following cumulative distribution function*

$$\begin{aligned} F(x) &= \tfrac{1+I(1/2,n/2,n/(n+x^2))}{2}, \quad x \geq 0, \qquad (1.4a) \\ F(x) &= 1 - F(-x), \quad x < 0, \qquad (1.4b) \end{aligned}$$

*with*

$$I(a, b, c) = \frac{1}{B(a, b)} \int_0^c t^{a-1}(1-t)^{b-1} dt, \qquad (1.5)$$

*where*

$$B(a, b) = \int_0^1 t^{a-1}(1-t)^{b-1} dt. \qquad (1.6)$$

*The parameter $n$ is called the degree of freedom of the Student's t-distribution.*

## 1.2 Markov process

Discrete and continuous random variables can model the effect of the outcomes of a random phenomenon on a given system. However, the value of a random variable is in general not sufficient to measure the impact of successive occurrences over time. In particular, we would like to observe the evolution of a system state that is subject to random variations. For this purpose, we define a **stochastic process**.

**Definition 1.2.1 (Stochastic process)**
*Let $\Omega$ be a sample space. A* ***stochastic process*** *$\{X(t), t \in \mathcal{T}\}$ is a collection of random variables defined on $\Omega$ and indexed with a parameter $t$. The set of all the possible values of the random variables is called the* ***state space*** *of the stochastic process, denoted by $\mathcal{S}$. It can be either discrete (countable) or continuous (uncountable). When the set of the parameter $t$ values, that is $\mathcal{T}$, is countable, the process is called a* ***discrete time*** *stochastic process. Otherwise, it is called a* ***continuous time*** *stochastic process.*

The parameter $t$ is often associated with a notion of time. Therefore, a stochastic process seems convenient for modelling the total effect of the successive occurrences of a random phenomenon. We can distinct four different types of stochastic process, depending on whether or not the state space $\mathcal{S}$ and the set $\mathcal{T}$ are countable. The following extension of Example 1.1.2 illustrates a discrete time stochastic process with a discrete state space.

**Example 1.2.1** *The price of a candy bag from RoyalJelly is determined only by the candies contained in the bag. It is equal to the sum of the price of the individual candies. When a customer comes in the shop, he gets an empty bag, puts a candy in it, then another one, and so on. For some sanitary reasons, a candy can never be removed from the bag. Thus, if we denote by $TotalPrice(i)$ the price of the bag after adding a $i$th candy into it and $c_j$ the colour of the $j$th added candy, we have that:*

$$TotalPrice(n) = \sum_{i=1}^{n} Price(c_i), \tag{1.7a}$$

$$TotalPrice(n) = TotalPrice(n-1) + Price(c_n). \tag{1.7b}$$

*The set $\{TotalPrice(n), n \in \mathbb{N}\}$ is a discrete time stochastic process defined on a countable state space, namely $\mathbb{N}$. It permits to observe the effect of successive outcomes of the "colour attribution" phenomenon on the total price of the candy bag.*

It is noteworthy that the stochastic process described in Example 1.2.1 is quite particular, because it has an additional property. Indeed, the new value of the bag after adding a candy is determined only by the price of this specific candy and the price of the bag before adding it. In other words, there is no need to know all the past prices to find out the new price. This property is called the **Markov property**. If it holds for a given stochastic process, this process is called a **Markov process**. Once again, we must make a clear distinction between Markov processes defined on a countable state space and the ones defined on a uncountable state space. Unless otherwise stated, we consider in this thesis only Markov processes with a countable state space, which are sometimes called **Markov chains**. To make their definition easier, we suggest to give first the following formal definition.

**Definition 1.2.2 (Conditional probability)**
*The **conditional probability** of the event $A$, given that the event $B$ occurs, noted $P(A|B)$ is defined as follows:*

$$P(A|B) = \frac{P(A \cap B)}{P(B)}. \tag{1.8}$$

Because a Markov process is a particular case of stochastic process, parameter $t$ takes its value either in a countable set or in an uncountable one. We separately formalize the definition of Markov process for both cases.

### 1.2.1 Discrete time Markov process

A Markov process is said to be in discrete time when the parameter $t$ indexing the random variables of the process takes its values in a countable set. According to Resnick and Sidney [27, Section 2.1], we formally define it as follows.

**Definition 1.2.3 (Discrete time Markov process)**
*A **discrete time Markov process** is a discrete stochastic process $\{X(t), t \in \mathcal{T}\}$, where $\mathcal{T}$ is **countable**, such that the Markov property holds. That is, $\forall x_1, x_2, ..., x_n \in \mathcal{S}, \forall n \in \mathcal{T}$, we have that*

$$\begin{aligned} P(X(n) = x_n | X(1) = x_1, X(2) = x_2, ..., X(n-1) = x_{n-1}) \\ = P(X(n) = x_n | X(n-1) = x_{n-1}). \end{aligned} \tag{1.9}$$

The Markov property allows to determine the next state of the Markov process by knowing only its current state, and ignoring the previous ones. A Markov process is defined by the probability distribution of the transitions between its states. In the discrete time case, a probability is associated with every transition. This means that, when in state $s$, the process can move to some other states with a given probability. Because the new state is modelled by the value of a random variable and according to Definition 1.1.4, the sum of the probability of all the transitions from $s$ must be equal to 1. For a discrete time Markov process, the set $\mathcal{T}$ can be seen as a set of epochs at which a transition occurs. Hence, for $t \in \mathcal{T}, X(t)$ denotes the state of the process after $t$ "steps"or "time units".

Usually, the probability of all the transitions of a Markov process are recorded in a stochastic matrix $P$ called the **transition matrix**. Basically, the element $P_{ij}$ of the matrix gives the probability of transiting from state $i$ to state $j$ in one step. Subsequently, if we denote by $\mathcal{S}$ the state space of the Markov process, we have:

$$\forall i, j \in \mathcal{S} : P_{ij} \geq 0, \tag{1.10a}$$

$$\forall i \in \mathcal{S} : \sum_{j \in \mathcal{S}} P_{ij} = 1. \tag{1.10b}$$

**Example 1.2.2** *In Example 1.2.1, we defined the discrete time Markov process $\{TotalPrice(n) \in \mathbb{N}, n \in \mathbb{N}\}$ to model the price evolution of a candy bag. In this case, the process can only move from state $s$ to state $s+1$, $s+2$ or $s+3$. Indeed, the price of a bag can only increase with respect to the successively added candies. Figure 1.2 shows the possible transitions in this Markov process. Every circle is a state and an arrow models a transition between two states. Each transition is labelled with its corresponding probability. The transition matrix of the process is given by*

$$P = \begin{pmatrix} 0 & 0.25 & 0.5 & 0.25 & 0 & 0 & \cdots \\ 0 & 0 & 0.25 & 0.5 & 0.25 & 0 & \cdots \\ 0 & 0 & 0 & 0.25 & 0.5 & 0.25 & \ddots \\ 0 & 0 & 0 & 0 & 0.25 & 0.5 & \ddots \\ 0 & 0 & 0 & 0 & 0 & 0.25 & \ddots \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \end{pmatrix}. \tag{1.11}$$

*Indeed, the price is increased by one unit if the newly added candy is yellow, by two units if it is green or orange, or three units if it is red. Furthermore, every kind of candy has a probability of 0.25 to be chosen.*

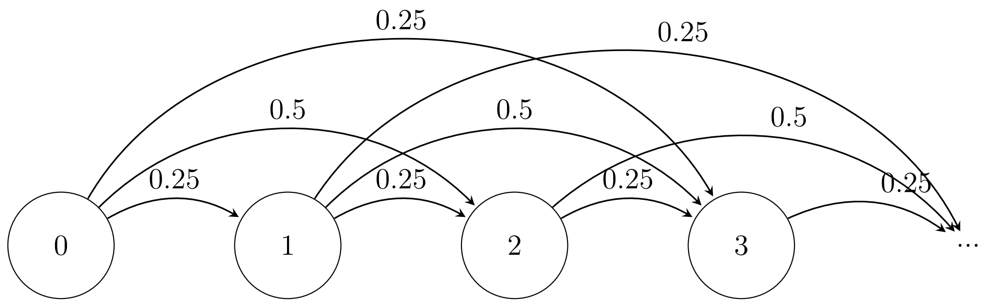


Figure 1.2: The possible transitions of the stochastic process presented in Example 1.2.1.

It is noteworthy that a transition matrix can be infinitely sized, similarly with the one presented in Equation (1.11).

### 1.2.2 Continuous time Markov process

We now define a continuous time Markov process, according to Resnick and Sidney [27, Section 5.1].

**Definition 1.2.4 (Continuous time Markov process)**
*A* ***continuous time Markov process*** *is a stochastic process* $\{X(t), t \in \mathcal{T}\}$*, where* $\mathcal{T}$ *is* ***uncountable****, such that the Markov property holds: that is, for any* $t, s \in \mathcal{T}$ *and any state* $j$*, we have*

$$P(X(t+s) = j | X(u), \forall u \in [0,t]) = P(X(t+s) = j | X(t)). \tag{1.12}$$

*Furthermore, the Markov process is* ***time-homogeneous*** *if, for all* $t, s \in \mathcal{T}, j \in S$*, it satisfies*

$$P(X(t+s) = j | X(t)) = P(X(s) = j | X(0)). \tag{1.13}$$

From now on, we always assume that a continuous time Markov process is time-homogeneous.

Instead of considering $\mathcal{T}$ as a set of transition steps, we can see it as an interval of time during which the state of the Markov process is observed. That is, for all $t \in \mathcal{T}, X(t)$ gives the process state observed at time $t$. A transition between two states occurs at random times. Therefore, we associate a rate of occurrence with every transition to express its tendency to occur. It is noteworthy that we have identically named both this tendency and the parameter of the exponential distribution. In fact, when a transition has a rate of occurrence equal to $\lambda$, it means that this transition can be triggered after a random time following an exponential distribution with parameter $\lambda$, hence the naming similarity.

Like the transition probabilities of a discrete time Markov process, the transition rates of a continuous time Markov process are usually recorded in a matrix. This matrix, denoted by $Q$, is called the **infinitesimal generator** of the process. If we denote by $\mathcal{S}$ the state space of the process, $Q$ is such that

$$\forall i \in \mathcal{S} : Q_{ii} = -\sum_{j \neq i} Q_{ij} \leq 0, \tag{1.14a}$$

$$\forall i, j \in \mathcal{S} : i \neq j \Rightarrow Q_{ij} \geq 0, \tag{1.14b}$$

$$\forall i \in \mathcal{S} : \sum_{j \in \mathcal{S}} Q_{ij} = 0. \tag{1.14c}$$

Basically, for $i \neq j$, the element $Q_{ij}$ encodes the transition rate from state $i$ to state $j$. $Q_{ii}$ is a negative rate. It expresses that, when the process reaches state $i$, it stays in this state for a random time exponentially distributed with rate $-Q_{ii}$. When this random time runs out, the process moves to state $j$, where $j \neq i$, with probability $\frac{-Q_{ij}}{Q_{ii}}$. Hence, a transition with a higher rate is more likely to occur. We now give an example of continuous time Markov process.

**Example 1.2.3** *We consider a Markov process $\mathcal{M} = \{X(t), t \in \mathbb{R}\}$ with state space $\{0, 1, 2\}$, whose transitions are shown in Figure 1.3. Every transition is labelled with its rate of occurrence. We assume that the process always starts in state 1. From state 1, two transitions are available: one to state 2 (with rate 4) and another one to state 0 (with rate 1). When in state 2, the process eventually reaches state 0 after a random time following an exponential distribution with rate 2. Once in state 0, the process never leaves it. Then, the infinitesimal generator of the process is a matrix*

$$Q = \begin{pmatrix} 0 & 0 & 0 \\ 1 & -5 & 4 \\ 2 & 0 & -2 \end{pmatrix}. \tag{1.15}$$

When a Markov process has no way to leave a given state (e.g. state 0 in Example 1.2.3), this state is called **absorbing**.

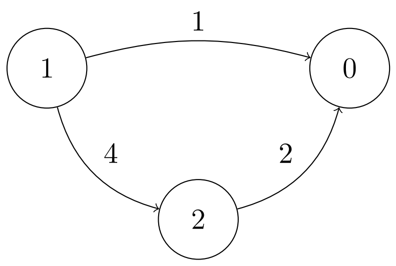


Figure 1.3: A continuous time Markov process with an absorbing state.

In Section 1.1.2, we gave a definition of continuous probability distribution (see Definition 1.1.6). We also introduced the exponential distribution (see Definition 1.1.7). Thanks to processes like the one specified in Example 1.2.4, we can define another continuous distribution, which is called the **phase-type distribution**. The definition we give below is based on Latouche and Ramaswami [18, Chapter 2].

**Definition 1.2.5 (Phase-type distribution)**
*Let us consider a continuous time Markov process defined on state space $\mathcal{S} = \{0, 1, \dots, m\}, m < \infty$ with initial probability vector $(\alpha_0, \boldsymbol{\alpha})$ and with infinitesimal generator*

$$Q = \begin{pmatrix} 0 & \boldsymbol{0} \\ \boldsymbol{t} & T \end{pmatrix}, \tag{1.16}$$

*where $\alpha$ is a row vector of size $m$, $T$ is a matrix of size $m \times m$, $t$ is a column vector of size $m$ and $\boldsymbol{0}$ is a row vector of size $m$ filled with zeros. We also assume that there is only one absorbing state. According to the properties of the infinitesimal generator of a continuous time Markov process (see Equation (1.14)), we have, $\forall i, j : 1 \leq i \neq j \leq m$, that*

$$T_{ii} < 0, \tag{1.17a}$$

$$T_{ij} \geq 0, \tag{1.17b}$$

$$-\sum_{k=1}^{m} T_{ik} = t_i \geq 0; \tag{1.17c}$$

*we also have that*

$$\alpha_0 + \sum_{k=1}^{m} \boldsymbol{\alpha}_k = 1. \tag{1.18}$$

*A continuous time **phase-type distribution** with parameters $\boldsymbol{\alpha}$ and $T$, denoted by $PH(\boldsymbol{\alpha}, T)$, is the distribution of time needed by the process to get absorbed into the absorbing state.*

Unless otherwise stated, we henceforth consider that $\alpha_0 = 0$. This implies that the process cannot enter the absorbing state immediately. We now give an example of a phase-type distribution based on the process described in Example 1.2.3.

**Example 1.2.4** *We consider the continuous time Markov process defined in Example 1.2.3. Then, if we assume that the process always starts in state 1, we can define a phase-type distribution $PH(\alpha, T)$ where*

$$\alpha = (1, 0), \tag{1.19a}$$

$$T = \begin{pmatrix} -5 & 4 \\ 0 & -2 \end{pmatrix}. \tag{1.19b}$$

Phase-type distributions are often used to model real processes that are composed of several operations. In this thesis, we make intensive use of them, especially in the case study carried out in Chapter 5.

Through Definition 1.2.1, Definition 1.2.3, and Definition 1.2.4, we introduced the Markov processes defined on a countable state space. The most common countable state spaces are subsets of $\mathbb{N}$. In this case, the state of a process is defined by an integer. However, it can be more convenient to model a system with a process whose states are defined by $n$-uplets, that is with a state space $\mathcal{S} \subseteq N^n$. In this case, the Markov process is said to be defined over a $n$-dimensional state space. We stress once again that we could use other n-dimensional state spaces. However, unless otherwise specified, we only consider subsets of $\mathbb{N}^n$. In particular, we are interested in a special case of two-dimensional Markov process, the **Quasi-Birth-and-Death** process.

## 1.3 Quasi-Birth-and-Death process

In this section, we present a particular case of Markov process, the **Quasi-Birth-and-Death**. Its particular features considerably simplify the structure of the matrix encoding its transitions. This introduction is based on Latouche and Ramaswami [18, Chapters 6,10,12]. The definition we give below concerns the continuous time QBDs, because the tool we ultimately want to build will work mainly with these ones. However, the definition of discrete time QBDs is very similar. Our main intent is to emphasize the particular features of such processes, which appear in both cases. Furthermore, there exist techniques that somehow convert a continuous time QBD into a discrete time one. We present these techniques further in this chapter.

### 1.3.1 Definition & Generator

We consider a Markov process $\{X(t), t \in \mathbb{R}^+\}$ defined over a two-dimensional state space

$$\{(k,i) : k \in \mathbb{N}, i \in \mathbb{N}, i \leq n_k \in \mathbb{N}\}.$$

We partition the state space into subsets

$$l(k) = \{(k,1), ..., (k,n_k)\} \tag{1.20}$$

called *levels*. In other words, level $k$ includes all states whose first coordinate is $k$. The second coordinate is often called the *phase* of the system. The number of states in level $k$, that is $n_k$, can be either finite or infinite. However, we consider from now on that this number is finite, unless otherwise stated.

Then, we apply a restriction on the transitions of the process, as illustrated in Figure 1.4. A transition from a state $(i,j)$ can only lead to another state of level $i$, or to a state of a directly adjacent level, that is $i+1$ (for any $i$) or $i-1$ (for $i > 0$). However, it cannot move in one step to a state of further or downer levels. Next, for any $i > 0$, we encode the transitions from the states of level $i$ to states of level $i-1$ in matrix $A_{-1}^{(i)}$, the transition between the states of level $i$ in matrix $A_0^{(i)}$ and the transitions from level $i$ states to level $i+1$ states in matrix $A_1^{(i)}$. We also record the transitions between states of level 0 in matrix $B_0$ and transitions from the states of level 0 to the states of level 1 in $B_1$. Subsequently, the infinitesimal generator of the process has the form

$$Q = \begin{pmatrix} B_0 & B_1 & 0 & 0 & \dots \\ A_{-1}^{(1)} & A_0^{(1)} & A_1^{(1)} & 0 & \dots \\ 0 & A_{-1}^{(2)} & A_0^{(2)} & A_1^{(2)} & \ddots \\ 0 & 0 & A_{-1}^{(3)} & A_0^{(3)} & \ddots \\ \vdots & \vdots & \ddots & \ddots & \ddots \end{pmatrix} \tag{1.21}$$

where $A_k^{(g)}$ is a $n_k \times n_{k+g}$ sized matrix, $g \in \{-1,0,1\}$, and $B_g$ is a $n_0 \times n_g$ sized matrix, with $g \in \{0,1\}$.

It is noteworthy that the generator of a QBD presents a particular structure: it is a block tridiagonal matrix (see Equation (1.21)). This particularity is essential because the computation of performance measures is based on the generator, as we will see later. It is also worth mentioning that according to our definition, the available transitions may depend on the current level. Such a QBD is called **level dependent** or **inhomogeneous**. If the available transitions and their respective rate are independent of the level,

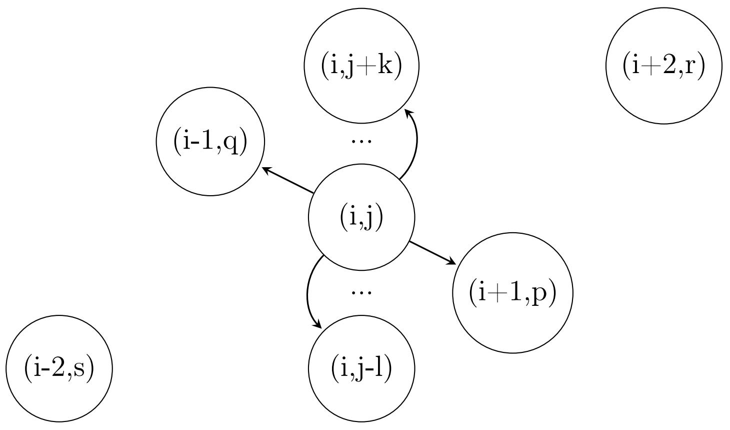


Figure 1.4: The available transitions in a Quasi-Birth-and-Death process.

level 0 excepted, the QBD is called **level independent** or **homogeneous**. In this case, we have

$$\forall i, j > 1, A_{-1}^{(i)} = A_{-1}^{(j)}, \tag{1.22}$$

$$\forall i, j > 0, A_0^{(i)} = A_0^{(j)} \wedge A_1^{(i)} = A_1^{(j)}. \tag{1.23}$$

Hence, the generator of a homogeneous QBD has the form

$$Q = \begin{pmatrix} B_0 & B_1 & 0 & 0 & \dots \\ B_{-1} & A_0 & A_1 & 0 & \dots \\ 0 & A_{-1} & A_0 & A_1 & \ddots \\ 0 & 0 & A_{-1} & A_0 & \ddots \\ \vdots & \vdots & \ddots & \ddots & \ddots \end{pmatrix} \tag{1.24}$$

where $A_j$ is a $m \times m$ matrix, $j \in \{-1, 0, 1\}$, with $m < \infty$ and $m \in \mathbb{N}$, and $B_0$ is a $k \times k$ matrix, with $k < \infty$ and $k \in \mathbb{N}$, $B_1$ is a $k \times m$ matrix and $B_{-1}$ is a $m \times k$ matrix.

It is worth mentioning that a QBD does not necessarily include an infinite number of levels. This number can indeed be finite. In this case, when in maximum level $M$, the process cannot move to level $M + 1$. Hence, the generator of an inhomogeneous and finite QBD has the form

$$Q = \begin{pmatrix} B_0 & B_1 & 0 & \dots & 0 \\ A_{-1}^{(1)} & A_0^{(1)} & A_1^{(1)} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & A_{-1}^{(M-1)} & A_0^{(M-1)} & A_1^{(M-1)} \\ 0 & \dots & \dots & C_{-1} & C_0 \end{pmatrix} \tag{1.25}$$

where $B_j, j \in \{0,1\}$ and $A_k^{(i)}, k\{0,1\}$ are defined as in Equation (1.21), and $C_l$, for $l \in \{-1,0\}$, is a $n_M \times n_{M+l}$ matrix.

If the QBD is homogeneous and finite, its generator has the form

$$Q = \begin{pmatrix} B_0 & B_1 & 0 & 0 & \dots & 0 \\ B_{-1} & A_0 & A_1 & 0 & \dots & \vdots \\ 0 & A_{-1} & A_0 & \ddots & \dots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & A_{-1} & A_0 & C_1 \\ 0 & \dots & \dots & 0 & C_{-1} & C_0 \end{pmatrix} \tag{1.26}$$

where matrices $B_j$ and $A_j$, for $j \in \{-1,0,1\}$, are defined as in Equation (1.24), and $C_1$ is a $m \times l$ matrix, $C_0$ is a $l \times l$ matrix and $C_{-1}$ is a $l \times m$ matrix, with $l < \infty$ and $l \in \mathbb{N}$.

Although the generator of a finite QBD is very similar to the generator of an infinite QBD, this feature is significant. Indeed, as we will see in Chapter 2, some algorithms must be modified or cannot be applied at all on a finite QBD.

We show next that a Quasi-Birth-and-Death is convenient to model some queueing systems. For recall, such a system receives jobs at random times, and completes them in a nondeterministic time.

### 1.3.2 The M/PH/1 queue

Thanks to the above definitions and concepts, we can already show that a Quasi-Birth-and-Death process can be used to model a specific queueing system. We consider an M/PH/1 queue, which is defined as follows. The interarrival times of jobs in the system follow an exponential distribution with rate $\lambda$. The system has one server that can complete one request at a time. The completion time is $PH(\alpha, T)$ distributed, where $T$ is a $m \times m$ sized matrix, $m \in \mathbb{N}$, with $m < \infty$. When a job enters the system, it waits in an infinitely sized queue if the server is already serving. The queue policy is *First In First Out (FIFO)*, that is a job cannot be served until all the other jobs that entered the system before it are completed. The server never idle, except when there is no more job in the system.

We consider the Markov process

$$\mathcal{M} = \{X(t), t \in \mathbb{R}^+\} \tag{1.27}$$

defined with the state space

$$\mathcal{S} = l(0) \cup l(1) \cup l(2) \cup \dots, \tag{1.28}$$

with

$$l(0) = \{0\}, \tag{1.29}$$

$$\forall i : 1 \leq i \in \mathbb{N} : l(i) = \{(i,1),(i,2),...,(i,m)\}. \tag{1.30}$$

When the $M/PH/1$ system is empty, the process is in state 0. Otherwise, the process is in state $(i, \phi)$ where $i$ is the total number of jobs in the system, and $\phi$ is the current state of the phase-type distribution related to the service currently performed, which we simply refer to as the service phase. Clearly, this Markov process is a QBD: from state $(i, \phi)$, it can only move to a state $(i', \phi')$ with $i' \in \{i-1, i, i+1\}$. Indeed, the state of the process is modified only when a new job enters the system $(i' = i+1)$ or when a service ends $(i' = i-1)$ or when only the service phase changes $(i' = i)$. Furthermore, the process has an additional property: apart from level 0, the possible transitions and their respective rate in a given level is independent of the level itself. Therefore, the same state variations can occur whatever the level. Indeed, independently of the number of jobs in the queueing system, a new request can arrive with a identical rate and the service time distribution never changes. Transitions from level 0 are a bit different because when in level 0, that is when the system is empty, there is no service currently being executed. Similarly, when the process transits from level 1 to level 0, that is when the only job in the system is completed, no new service must be started. Hence, the QBD is homogeneous. Furthermore, because the queue capacity is infinite, the QBD is also infinite. Its generator has therefore the form given in Equation (1.24). We now give an example of M/PH/1 queueing system and we define a QBD modelling it.

**Example 1.3.1** *Figure 1.5 presents the possible transitions of a QBD modelling an M/PH/1 queueing system that has an infinite capacity and the following parameters:*

$$\lambda = 0.5$$
$$\alpha = \begin{bmatrix} 1 & 0 \end{bmatrix}$$
$$T = \begin{bmatrix} -10 & 4 \\ 0 & -3 \end{bmatrix}$$

*where $\lambda$ is the rate of the exponential distribution followed by the interarrival times, and $PH(\alpha, T)$ is the phase-type distribution followed by the service times.*

*We now describe the content of each inner block of the generator defined in Equation (1.24) for the M/PH/1 queueing system illustrated in Figure 1.5. We observe that a repetitive structure appears, hence the homogeneous*

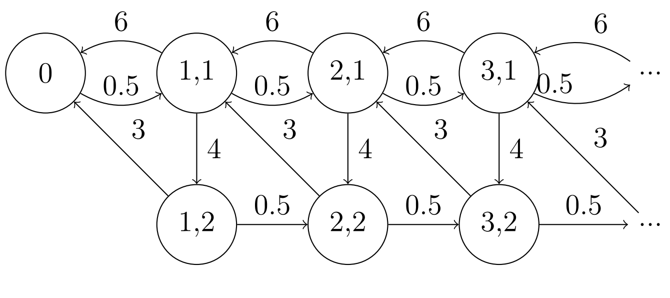


Figure 1.5: The possible transitions of a QBD modelling an M/PH/1 queueing system.

*features of the QBD. When the system is empty, that is when the QBD is in level* 0*, the arrival of a job is the only event that modifies the state of the system. When it happens, the job is served and enters either service phase 1 with probability* $\alpha_1$*, or service phase 2 with probability* $\alpha_2$*. Therefore, when leaving state 0, the process can only reach state* $(1, 1)$ *with rate* $\lambda\alpha_1$ *or state* $(1, 2)$ *with rate* $\lambda\alpha_2$*. Hence, because of the generator properties described in Equation (1.14), we have that*

$$B_0 = \quad -(\lambda\alpha_1 + \lambda\alpha_2) \quad = -\lambda \tag{1.31}$$

$$B_1 = \quad (\lambda\alpha_1, \lambda\alpha_2) \quad = \lambda\alpha \tag{1.32}$$

*When only one job is in the system, the process is in level* 1*. Provided that no other request has been received meanwhile, it moves to level* 0 *upon the job completion, that is when the phase-type distribution defining the service time enters its absorbing state (see Definition 1.2.5). Hence, the transition rates from level* 1 *to level* 0 *are given by*

$$B_{-1} = -T\boldsymbol{1} = \boldsymbol{t} \tag{1.33}$$

*where* ***1*** *is a column vector of appropriate size, filled with ones.*

*When the process is in level* $i$*, with* $i > 1$*, it also moves to level* $i - 1$ *upon the end of a service. However, because there are still jobs to complete, a new service starts. Subsequently, the newly started phase-type distribution must be taken into account. The beginning phase is determined with respect to the probability vector* $\alpha$*. Then, we have*

$$A_{-1} = \boldsymbol{t}\alpha. \tag{1.34}$$

*Only matrices* $A_0$ *and* $A_1$ *have still to be specified. When the process is in level* $i > 0$*, there exist transitions that lead to another state of this same level. These transitions correspond to the transitions between two phases of the service process. Indeed, moving from one service phase to another does* *not impact on the total number of jobs in the system. The rates of these transitions are recorded in matrix $T$. Independently of the service phase, an arrival can occur. This event increases the number of jobs in the system but has no influence on the service phase. Hence, because of the properties of a generator defined in Equation (1.14), we have that*

$$A_1 = \lambda I, \tag{1.35}$$

$$A_0 = T - A_1, \tag{1.36}$$

*where $I$ is an identity matrix with the same size as $T$.*

We have thus defined the complete generator of a QBD modelling an $M/PH/1$ queueing system. Basically, most of the queueing systems with one arrival at a time can be modelled with a QBD. The Quasi-Birth-and-Death process seems to be a sufficiently expressive type of model. Their analysis is therefore of interest. In Chapters 2 and 3, we present different methods to compute performance measures. The algorithms computing these measures make intensive use of the particular structure of the QBD generator, as we will show. However, some of them require a transition matrix instead of a generator, hence the need of a method to transform a continuous time QBD into a discrete time one.

### 1.3.3 Uniformization of Markov processes

We previously mentioned that a continuous time Quasi-Birth-and-Death can be somehow transformed into in discrete time one. In this subsection, we present a well-known technique that transforms the generator of a continuous time Markov process into a transition matrix. It is called the **uniformization technique**. In Chapter 2, we motivate the need for this technique, because we will mainly reason about discrete time processes. Furthermore, the uniformization plays an essential role in the simulation algorithms we develop in Chapter 3. This introduction is given according to Latouche and Ramaswami [18, Section 2.8], and to Resnick and Sidney [27, Section 5.10].

We consider a continuous time Markov process $\mathcal{M} = \{X(t), t \in \mathbb{R}^+\}$ with countable state space $\mathcal{S}$ and generator $Q$. Let $c$ be a real number such that

$$\forall i \in \mathcal{S} : |Q_{ii}| \leq c < \infty. \tag{1.37}$$

Then, we define the matrix $K$ as

$$K = \frac{1}{c}Q + I \tag{1.38}$$

where $I$ is the identity matrix of the same size as Q. From Equations (1.14) we have that

$$\begin{aligned}
\forall i \in \mathcal{S}: \quad 0 \leq K_{ii} &= \frac{1}{c} Q_{ii} + 1 \leq 1, && (1.39a)\\
\forall i,j \in \mathcal{S}: \quad 0 \leq K_{ij} &= \frac{1}{c} Q_{ij} \leq 1, && (1.39b)\\
\forall i \in \mathcal{S}: \quad \sum_{j \in \mathcal{S}} K_{ij} &= 1 + \frac{1}{c} \sum_{j \in \mathcal{S}} Q_{ij} = 1. && (1.39c)
\end{aligned}$$

$K$ is therefore a stochastic matrix (see Equation (1.10)).

Let $\mathcal{M}' = \{Y(n), n \in \mathbb{N}\}$ be a discrete time Markov process with transition matrix $K$. We also consider an infinite sequence of independent events such that the time between two events follows an exponential distribution with rate $c$. We denote by $p_n$, with $n \in \mathbb{N}$ the occurrence time of the n$^{\text{th}}$ event and we define the stochastic process

$$\mathcal{M}'' = \{Z(t), t \in \mathbb{R}^+ : Z(t) = Y(n), n \in \mathbb{N}, t \in [p_n, p_n + 1]\} \tag{1.40}$$

which is markovian. Its infinitesimal generator is equal to $Q$, as proven by Latouche and Ramaswami [18, Section 2.8]. Intuitively, the uniformization means that the process $\mathcal{M}$ can move to another state only at times $p_n, n \in \mathbb{N}$. Provided that the process is currently in state $i$, it moves to state $j$ with probability $K_{ij}$.

Finally, it is noteworthy that the specific value of $c$ has no importance, as long as it is finite and greater than the absolute value of any diagonal element of $Q$. Usually, it is defined as

$$c = \max_{i \in \mathcal{S}} (|Q_{ii}|). \tag{1.41}$$

The choice of this value can have significant consequences, in particular on the efficiency of the simulation algorithms we present in Chapter 3.

We discuss next how the uniformization technique can be specifically applied to Quasi-Birth-and-Death processes. The diagonal elements of an inhomogeneous QBD generator are contained in matrices $B_0$ and $A_0^{(i)}$ (see Equation (1.21)). If the QBD is homogeneous, we have that

$$\forall i,j \in \mathbb{N}_0, A_0^{(i)} = A_0^{(j)} = A_0 \tag{1.42}$$

(see Equation (1.24)). If it is also infinite, we define the uniformization rate as

$$c \geq \max( \max_{i \in [1..k]} (|(B_0)_{ii}|), \max_{i \in [1..m]} (|(A_0)_{ii}|)) \tag{1.43}$$

where $k$ and $m$ are respectively the size of $B_0$ and the size of $A_0$. On the other hand, if the QBD is finite, we must also take into account matrix $C_0$, which records the inner transitions of the maximum level (see Equation (1.26)). In this case, the uniformization rate becomes

$$c' \geq \max(c, \max_{i\in[1..l]}(|(C_0)_{ii}|)) \tag{1.44}$$

where $c$ is defined as in Equation (1.43) and $l$ is the size of $C_0$.

When the QBD is inhomogeneous, the computation is hardened. If the QBD is finite with maximum level $M$ (see Equation (1.25)), the number of required comparisons dramatically increases. Indeed, the uniformization rate $c$ is given by

$$c \geq \max(c_0, c_M, c_k) \tag{1.45}$$

with

$$c_0 = \max_{i\in[1..n_0]}(|(B_0)_{ii}|), \tag{1.46a}$$

$$c_M = \max_{i\in[1..n_M]}(|(C_0)_{ii}|), \tag{1.46b}$$

$$c_k = \max_{i\in[1..M-1],j\in[1..n_i]}(|(A_0^{(i)})_{jj}|), \tag{1.46c}$$

where $n_0$ is the size of $B_0$, $n_i$, with $0 < i < M$, is the size of $A_0^{(i)}$, and $n_M$ is the size of $C_0$. Finally, if the QBD is inhomogeneous and infinite, $c$ is given by

$$c \geq \max(c_0, c_k) \tag{1.47}$$

with

$$c_0 = \max_{i\in[1..n_0]}(|(B_0)_{ii}|), \tag{1.48a}$$

$$c_k = \max_{i\in\mathbb{N}_0,j\in[1..n_i]}(|(A_0^{(i)})_{jj}|). \tag{1.48b}$$

It is noteworthy that in Equation (1.48b), $c_k$ is defined as the maximum element of an infinite set. In an implemented program, we cannot determine the maximum value of an infinite set whose contained values are a priori unknown, because it would require an infinite computation time. Some assumptions about the process can help to overcome this issue. We do not discuss this point, though.

We illustrate the use of the uniformization technique on the infinite and homogeneous QBD presented in Example 1.2.4.

**Example 1.3.2** *Let us consider the Quasi-Birth-Death process defined in Example 1.3.1. In order to uniformize it, we have to determine a real value c greater or equal to the absolute value of any diagonal element. Because the QBD is homogeneous and infinite, according to Equation (1.43), we can define c as:*

$$\begin{aligned} c &= \max(|-\lambda|, \max_{i\in[1..2]}(|T_{ii} - \lambda|)) \\ &= \max(0.5, \max(10.5, 3.5)) \\ &= 10.5 \end{aligned} \tag{1.49}$$

*Then, if we denote by Q the infinitesimal generator of the QBD, we define the transition matrix of the discrete time version of the process as*

$$K = I + \frac{1}{c}Q = \begin{pmatrix} B'_0 & B'_1 & 0 & 0 & \dots \\ B'_{-1} & A'_0 & A'_1 & 0 & \dots \\ 0 & A'_{-1} & A'_0 & A'_1 & \ddots \\ 0 & 0 & A'_{-1} & A'_0 & \ddots \\ \vdots & \vdots & \ddots & \ddots & \ddots \end{pmatrix} \tag{1.50}$$

*where*

$$B'_0 = 1 + \tfrac{-lambda}{c} = \left(\tfrac{20}{21}\right), \tag{1.51a}$$

$$B'_1 = \tfrac{lambda}{c}\alpha = \left(\tfrac{1}{21} \quad 0\right), \tag{1.51b}$$

$$B'_{-1} = \tfrac{1}{c}t = \begin{pmatrix} \frac{12}{21} \\ \frac{6}{21} \end{pmatrix}, \tag{1.51c}$$

$$A'_0 = \tfrac{1}{c}(I + (T - \lambda I)) = \begin{pmatrix} 0 & \frac{8}{21} \\ 0 & \frac{14}{21} \end{pmatrix}, \tag{1.51d}$$

$$A'_1 = \tfrac{1}{c}\lambda I = \begin{pmatrix} \frac{1}{21} & 0 \\ 0 & \frac{1}{21} \end{pmatrix}, \tag{1.51e}$$

$$A'_{-1} = \tfrac{1}{c}t\alpha = \begin{pmatrix} \frac{12}{21} & 0 \\ \frac{6}{21} & 0 \end{pmatrix}. \tag{1.51f}$$

### 1.3.4 Kendall's notation

We end this chapter by presenting a notation we use to describe the features of a queueing system. The notation was introduced by D. G. Kendall in 1953 [32], and it bears its creator's name.

**Definition 1.3.1 (Kendall's notation)**
*Kendall's notation is a concise description of a queueing system. The template of the notation is* ***A/B/C/D/E/F*** *where*

- ***A*** *is a code specifying the probability distribution of the interarrival times,*
- ***B*** *is a code defining the distribution of a service time,*
- ***C*** *is an integer giving the number of servers,*
- ***D*** *is an integer specifying the queue capacity,*
- ***E*** *is the total number of jobs that can enter in the system, and*
- ***F*** *gives the queue discipline.*

*There exists a more concise form of the Kendall's notation. Its template is* ***A/B/C****. When that form is used, it is assumed that $K$ is $\infty$, $N$ is $\infty$ and $D$ is $FIFO$ (First In First Out).*

In particular, we are interested in two codes commonly used to describe a time distribution: **M** and **PH**. The **M** code stands for "Markovian". It defines a random time that follows an *exponential distribution*. The **PH** code denotes a phase-type distribution. We already used both codes to describe an M/PH/1 queueing system.

## 1.4 Conclusion

In this section, we have given a formal definition of the Quasi-Birth-and-Death process. We also have put emphasis on its particular features, especially the form of its infinitesimal generator. We have presented a technique called uniformization that converts a continuous time QBD into a discrete time QBD. Finally, we have introduced the Kendall's notation to specify a queueing system.

In the next chapter, we present theoretical measures that are relevant for the performance analysis of a system. We also give algorithms to compute these measures when the system is modelled by a Quasi-Birth-and-Death process and we discuss the time complexity of the presented algorithms.

# Part II

# Computation Methods

# Chapter 2

# Matrix Analytic Methods

In Chapter 1, we described the Quasi-Birth-and-Death process as a convenient mathematical model for systems whose state depends on the outcome of random phenomena. Our next objective is to present methods to evaluate the performance of such a system. When the system is modelled by a Markov process, the performance evaluation often consists in the determination of so-called measures of performance.

In this chapter, we especially put emphasis on two specific measures: the stationary distribution and the first passage times. In Section 2.1, we first define the stationary distribution as well as the conditions for it to exist. Next, we present methods to compute it, which are based on mathematical relations specific to QBDs. These methods, called *matrix analytic methods*, make intensively use of the particular structure of the QBD. They allow the development of several algorithms to determine the stationary distribution. In particular, Subsection 2.1.3 is dedicated to the computation of the stationary vector when the QBD is homogeneous and infinite. Subsection 2.1.4 deals with the other types of QBDs. Next, in Section 2.2, we focus on the second measure, that is the first passage times. Again, we present methods which serve as a basis for the development of algorithms to compute them.

## 2.1 Stationary Distribution

The first performance measure we focus on is called **stationary distribution**. This measure characterizes the so-called **steady state** of a Markov process. In simple terms, the steady state can be described as the long-run behaviour of the considered process. On the contrary, the short-run behaviour of the process is referred to as **transient state**.

The stationary distribution is described by probability values recorded in a vector called either **stationary probability vector** or steady state probability vector. Intuitively, this vector records the probability of the process being in a given state after a long run time.

First, we give the conditions for the stationary distribution to exist in Subsection 2.1.1. Then, we give an algorithm that computes the stationary vector of any Markov process in Subsection 2.1.2. Next, we make use of the particular structure of the Quasi-Birth-Death process to develop algorithms that compute the stationary vector of such a process. More precisely, we present in Subsection 2.1.3 the *Logarithmic Reduction* algorithm, which can be applied to homogeneous and infinite QBDs. In order to deal with the other types of QBD, we present the *Linear Level Reduction* algorithm in Subsection 2.1.4.

### 2.1.1 Stability conditions

Many performance measures are computed thanks to the stationary distribution of the Markov process that models the considered system. However, this distribution does not always exist. Before studying algorithms to compute it, we discuss the conditions in which the stationary distribution exists and is independent of the initial state. To give these conditions, different definitions are needed. More precisely, we define several properties related to Markov processes and we establish the link between these properties and the stationary distribution. The definitions of these properties we give are based on Resnick and Sidney [27, Chapter 2]. For the sake of readability, we always denote the state space of a Markov process by $\mathcal{S}$ and we refer to the set of values of the time parameter as $\mathcal{T}$. We also assume that $\mathcal{S}$ is always countable. Furthermore, $\mathcal{T}$ is countable for a discrete time process and it is uncountable for a continuous time process.

The first property we focus on concerns only discrete time Markov processes. It is called **period**. Intuitively, a state is said to have period $p$ if he can only be reached after a number of time units that is a multiple of $p$. We formally define periodicity and we illustrate it with an example.

**Definition 2.1.1 (Period)**
*For $i \in \mathcal{S}$, state $i$ is said to have period $p$ if, starting from it, the number of steps needed to come back to it must necessarily be a multiple of $p$. More formally,*

$$\forall t \in \mathcal{T} : P(X(t) = i | X(0) = i) > 0 \implies t = cp, c \in \mathbb{N}. \tag{2.1}$$

*If a state has period 1, then it is called **aperiodic**. Otherwise, it is called **periodic**.*

**Example 2.1.1** *Let $\mathcal{M} = \{X(t), t \in \mathbb{N}\}$ be a discrete time Markov process with state space $\mathcal{S} = \{1, 2\}$ characterized by the following transition matrix*

$$P = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{2.2}$$

*Obviously, if the process always starts in state 1, that is*

$$P(X(0) = 1) = 1,$$

*the process can only be in state 1 at odd time units, while it can only be in state 2 even time units. Therefore, the period of both state is 2.*

The next property we present concerns both discrete time and continuous time Markov processes. The **recurrence** of a state raises the possibility to reach a state infinitely often. We formally define it as follows.

**Definition 2.1.2 (Recurrent state)**
*If, from state $i$, the process eventually reaches state $i$ again in the future with probability $f_{ii}$, then state $i$ is said to be* ***recurrent*** *if and only if $f_{ii} = 1$.*

There exist two kinds of recurrence, which are defined as follows.

**Definition 2.1.3 (Positive & Null recurrent state)**
*Let $\mu_i$ be the mean recurrence time of a recurrent state $i$, that is the expected time for the process, starting from state $i$, to be in $i$ again in the future. Then, state $i$ is said to be* ***recurrent positive*** *if $\mu_i < \infty$. Otherwise, it is called* ***null recurrent****.*

The periodicity and the recurrence properties individually characterize the states of a process. We want to extend such properties to the whole state space. In order to achieve this, we first study some communication relations between the states.

**Definition 2.1.4 (Accessible state)**
*State $j$ is* ***accessible*** *from state $i$, noted $i \to j$, if and only if there is a path from $i$ to $j$, that is*

$$\exists t \in \mathcal{T}, t < \infty : P(X(t) = j | X(0) = i) > 0. \tag{2.3}$$

**Definition 2.1.5 (Communicating state)**
*State $i$ and state $j$ **communicate**, noted $i \leftrightarrow j$, if and only if $i \rightarrow j$ and $i \leftarrow j$.*

Thanks to the above definitions, we can separate the states in groups such that a state of a given group communicates with every state of the same group. Such a group is called equivalence class.

**Definition 2.1.6 (Equivalence class)**
*An **equivalence class** $C \subseteq \mathcal{S}$ is a set of states such that*

$$\forall i, j \in C, \forall k \in \mathcal{S} \setminus C : i \leftrightarrow j \wedge (k \not\leftrightarrow i) \tag{2.4}$$

The state space of a Markov process can be partitioned into equivalence classes. Indeed, a state that communicates with no other state forms an equivalence class, and according to our definition, two states that communicate are necessarily part of the same equivalence class. The concept of equivalence class allows to infer the periodicity and the recurrence property of the whole equivalence class from the property of a single state of this class, as established by the following theorem.

**Theorem 2.1.1 (Homogeneity in an equivalence class)**
*All the states of a given equivalence class are either all non recurrent, all positive recurrent, or all null recurrent. Furthermore, they all have the same period.*

We can therefore characterize a class as non recurrent, positive recurrent or null recurrent. In particular, if the state space of a process is composed of only one equivalence class, the same recurrence property holds for all of the states. Such a process is called **irreductible**. Depending on the recurrence property of its states, the process is also called either non recurrent, positive recurrent, or null recurrent. Furthermore, if the states of an irreductible process are aperiodic, the process is also called aperiodic.

The properties we have introduced above allow us to specify the conditions for the stationary distribution of a process to exist. These conditions are given in the following two theorems.

**Theorem 2.1.2 (Stability conditions (discrete time))**
*If a discrete time Markov process is aperiodic, irreductible, and positive recurrent, its stationary distribution exists and is independent of the initial probability distribution. Furthermore, the stationary probability vector is the unique solution of the system of equations*

$$\pi P = \pi \tag{2.5a}$$
$$\pi \boldsymbol{1} = 1 \tag{2.5b}$$

*where $P$ is the transition matrix associated with the process and $\boldsymbol{1}$ is a vector of appropriate size filled with ones.*

**Theorem 2.1.3 (Stability conditions (continuous time))**
*If a continuous time Markov process is irreductible and positive recurrent, its stationary distribution exists and is independent of the initial probability distribution. Furthermore, the stationary probability vector is the unique solution of the system of equations*

$$\pi Q = 0 \tag{2.6a}$$
$$\pi \boldsymbol{1} = 1 \tag{2.6b}$$

*where $Q$ is the infinitesimal generator of the process and $\boldsymbol{1}$ is a vector of appropriate size filled with ones.*

We henceforth consider Markov processes which meets the conditions for having a stationary distribution. We also assume, for the rest of the chapter, that the processes we work with are discrete. We discuss the continuous case at the end of the chapter.

### 2.1.2 Gaussian Elimination algorithm

Thanks to Equations (2.5), we can already develop a method for computing the stationary probability vector of a Markov process. The method consists in determining the solution of Equations (2.5). Obviously, it cannot be applied when the state space of the considered process is infinite[1], that is when both $\pi$ and $P$ are infinitely sized. It would result in an infinite number of

[1]Be aware that a given set can be countable but infinite.

computations. Hence, the run time of a program performing such computations would be infinite.

Latouche and Ramaswami [18, Section 5.4] present an algorithm to solve this system of equations which is based on Gaussian elimination and LU decomposition. We refer to Press *et al.* [25, Section 2.3] for a clear introduction to Gaussian elimination and LU decomposition. We give an implementation of this algorithm in an OCTAVE function whose code is shown in Figure 2.1. It is noteworthy that we choose to give a real implementation instead of pseudocode. Actually, the tool we build uses OCTAVE functions to compute the performance measures. Hence, we present every algorithm in the form of such a function.

The function has only one parameter, which is the transition matrix $P$, and it returns vector $\pi$. We first compute matrix L of the $LU$ decomposition (lines 2-14). Next, we compute vector $\pi$ by back-substitution (lines 15-23) and we normalize it (lines 24-27).

**Proposition 2.1.1**
*The Gaussian elimination has a worst-case time complexity of $\mathcal{O}(n^3)$ in the number of scalar operations, where $n$ is the number of rows and columns of the matrix given as parameter.*

Indeed, the first part of the algorithm (lines 5-14) executes three embedded loops. The inner loop (lines 10-12) performs a scalar addition at most $n$ times. It is repeated at most $n$ times in the middle loop (lines 8-13). Finally, the algorithm iterates in the outer loop $n-1$ times.

### 2.1.3 Logarithmic Reduction algorithm

Instead of using the Gaussian elimination algorithm to compute the stationary vector of a QBD, we present more efficient, QBD-specific algorithms that profit from the particular structure of this kind of Markov process. These ones are based on matrix analytic methods. We aim to provide intuitive reasoning over these methods in order to understand the underlying theoretical foundations essential for the understanding of the presented algorithms. We also evaluate the efficiency of these algorithms by determining their worst-case time complexity.

```
function pi = GLU(P)
        # Computes matrix L of the decomposition P - I = LU
        A = P;
        n = size(P,1);
        for i = 1:(n-1)
                s = sum(A(i,(i+1):n));
                A(i,i) = s;
                for k = (i+1):n
                        t = A(k,i)/s;
                        for j = (i+1):n
                                A(k,j) = A(k,j) + t * A(i,j);
                        end
                end
        end
        # Computes pi by back-substitution
        pi(n) = 1;
        for j = (n-1):(-1):1
                s = 0;
                for i = (j+1):n
                        s = s + pi(i) * A(i,j);
                end
                pi(j) = s / A(j,j);
        end
        s = 0;
        # Normalizes pi
        s = sum(pi);
        pi = pi / s;
endfunction
```

Figure 2.1: An OCTAVE function that solves the system $\pi = \pi P, \pi\mathbf{1} = 1$, based on Gaussian elimination.

The development of an algorithm to compute the stationary vector of a QBD highly depends on the structure of the QBD itself. In Section 1.3.1, we stated that a QBD can be either homogeneous or inhomogeneous. We also distinguished whether or not the number of levels is infinite. Depending on these features, different algorithms can be applied. We choose to adopt a structure similar to Latouche and Ramaswami [18], because it is more appropriate when the focus is on the probabilistic interpretation. First, we establish in this Subsection relations that hold when the QBD is homogeneous and infinite, and we develop a new algorithm based on these relations.

### Determination of $\pi_0$ and $\pi_1$

We consider a discrete time, homogeneous and finite QBD $\{X(t), t \in \mathbb{N}\}$ defined on state space

$$\mathcal{S} = \{(k,i) : 1 \leq i \leq n\} \subset \mathbb{N}^2. \tag{2.7}$$

We want to achieve the computation of its stationary vector $\pi$, that is the unique solution of Equations (2.5). We divide $\pi$ into subvectors $\pi_k, k \in \mathbb{N}$, such that

$$\pi_k = \pi_{k1}, ..., \pi_{kn} \tag{2.8}$$

records the stationary probability of states $(k, 1)$, ..., $(k, n)$ respectively. Then, Equations (2.5) can be rewritten as follows:

$$\pi_0 B_0 + \pi_1 B_{-1} = \pi_0 \tag{2.9a}$$
$$\pi_0 B_1 + \pi_1 A_0 + \pi_2 A_{-1} = \pi_1 \tag{2.9b}$$
$$\pi_{k-1} A_1 + \pi_k A_0 + \pi_{k+1} A_{-1} = \pi_k \tag{2.9c}$$
$$\sum_{j \geq 0} \pi_j \mathbf{1} = 1 \tag{2.9d}$$

with $k > 1$.

Thanks to this system of equations combined with the following theorem, we can explicitly compute any subvector $\pi_k$. We refer to Latouche and Ramaswami [18, Section 6.2] for a formal proof of the theorem.

**Theorem 2.1.4 (Rate matrix existence)**
*If an infinite and homogeneous QBD is positive recurrent, then there exists a non negative matrix $R$ of order $n$, called the **rate matrix**, such that*

$$\forall k > 0 : \pi_{k+1} = \pi_k R \tag{2.10}$$

*which can also be written as*

$$\forall k > 0 : \pi_{k+1} = \pi_1 R^k \tag{2.11}$$

Matrix $R$ is such that, for $0 < i, j < n$ and $k > 0$, $R_{ij}$ is the expected number of visits to state $(k + 1, j)$ before a return to any state of levels 0 to $k$, given that the process starts in $(k, i)$. In other words, $R_{ij}$ denotes the expected time spent in $(k + 1, j)$ per time unit spent in $(k, i)$. We can therefore give an interpretation of Equation (2.10). Because $\pi_{ki}$ denotes the expected time spent in state $(k, i)$ per time unit of the global process, $\pi_{ki} R_{ij}$ is obviously the expected time spent in $(k+1, j)$ per global time unit, which is equal to $\pi_{(k+1)j}$.

According to the above theorem and Equations (2.9), we can compute the whole vector $\pi$ thanks to the rate matrix and the inner blocks of the transition matrix. First, we give an expression to compute $\pi_0$ and $\pi_1$. For this purpose, we consider Equations (2.9a) and (2.9b). In (2.9b), we replace $\pi_2$ by $\pi_1 R$, according to Equation (2.10). Then, we have that

$$\pi_0 B_0 + \pi_1 B_{-1} = \pi_0, \tag{2.12a}$$

$$\pi_0 B_1 + \pi_1 A_0 + \pi_1 R A_{-1} = \pi_1. \tag{2.12b}$$

According to this system of equations and Equation (2.9d), we conclude that $(\pi_0, \pi_1)$ is the unique solution of

$$\begin{aligned}(\pi_0, \pi_1) &= \big(\pi_0 B_0 + \pi_1 B_{-1}, \pi_0 B_1 + \pi_1 A_0 + \pi_1 R A_{-1}\big) \\ &= (\pi_0, \pi_1) \begin{pmatrix} B_0 & B_1 \\ B_{-1} & A_0 + R * A_{-1} \end{pmatrix}\end{aligned} \tag{2.13}$$

and we normalize $\pi_0$ and $\pi_1$ with

$$\pi_0 \mathbf{1} + \pi_1 \sum_{k \geq 0} R^k \mathbf{1} = 1. \tag{2.14}$$

Equation (2.13) is directly obtained from Equations (2.9a) and (2.9b). Equation (2.14) normalizes vector $(\pi_0, \pi_1)$ and is obtained from Equations (2.9d) and (2.11).

To avoid successively computing the powers of $R$, we must determine how the series

$$\sum_{k>0} R^k \tag{2.15}$$

converges. Before giving a theorem that establishes this convergence, we need the following definition, according to Gradshteyn and Ryzhik [10, Section 15.51].

**Definition 2.1.7 (Spectral radius)**
*Let $M$ be a $m \times m$ sized matrix with real elements. Let $\lambda_1, ..., \lambda_m$ be the eigenvalues of $M$. Then, the **spectral radius** of $M$, noted $sp(M)$, is defined as*

$$sp(M) = \max_{1 \leq i \leq m} |\lambda_i| \tag{2.16}$$

The link between the convergence of the series (2.15) and the spectral radius of $R$ is established by the following theorem [18, Section 5.1].

**Theorem 2.1.5 (Convergence of the series $\sum_{k\geq 0} M^k$)**
*Let $M$ be a matrix of finite order. The series*

$$\sum_{k\geq 0} M^k$$

*converges to*

$$(I - M)^{-1}$$

*is and only if $sp(M) < 1$.*

Thanks to this theorem, we infer that Equation (2.14) is equivalent to

$$\pi_0 \mathbf{1} + \pi_1 (I - R)^{-1} \mathbf{1} = 1. \tag{2.17}$$

We can therefore compute vector $(\pi_0, \pi_1)$ by using an equation solving algorithm that is similar to the Gaussian elimination we described in Section 2.1.2 and by normalizing with Equation (2.17). Thanks to the rate matrix and $\pi_1$, any subvector $\pi_k$, $k > 1$, can then be computed by solving Equation (2.10). The key point for determining the whole stationary vector is therefore the computation of $R$.

### Properties of the rate matrix

The method to compute the rate matrix we present relies on the relations that holds between $R$ and two other matrices we denote by $G$ and $U$. In particular, we show that whenever one of the three matrices is known, the other two can be determined. Hence, computing $R$ comes to developing an algorithm that determines either $G$ or $U$. In this section, we present the relationship between these three matrices and we give a probabilistic interpretation of these relations. We refer to Latouche and Ramaswami [18, Chapter 6] for a formal proof of the properties we give.

First, we define $U$ as the matrix recording, for any $k > 1$, the probability of return to level $k$ before a visit in level $k-1$, given that the process starts in level $k$. In other words, $U_{ij}$ encodes the probability of reaching a state of level $k$, which is precisely $(k, j)$, before reaching level $k-1$, given that the process starts in $(k, i)$. Obviously, in order to reach $(k, j)$ from $(k, i)$ without visiting any state of level $k-1$ and any other state of level $k$ meanwhile, either

1. the process stays in the same level and moves immediately from phase $i$ to phase $j$, or

2. it moves to level $k+1$ before eventually going down from a state $(k+1, h)$ to $(k, j)$.

In the former case, we know that the probability of transiting in one step from $(k, i)$ to $(k, j)$ is $(A_0)_{ij}$. In the latter case, from the definition of $R$, we know that (k+1,h) is expected to be visited $R_{ih}$ times. Furthermore, the probability to move from $(k+1, h)$ to $(k, j)$ is $(A_{-1})_{hj}$ and state $(k, j)$ can be reached from any state of level $k+1$. Hence, we have that

$$U_{ij} = (A_0)_{ij} + \sum_{1 \leq h \leq n} R_{ih}(A_{-1})_{hj} \tag{2.18}$$

which can also be written as

$$U = A_0 + RA_{-1}. \tag{2.19}$$

It is worth mentioning that $(U^n)_{ij}$ is the probability, starting in $(k, i)$, of visiting level $k$ for the $n^{\text{th}}$ time, precisely in state $(k, j)$, before reaching the lower level. Next, we define matrix $N$ as

$$N = \sum_{i \geq 0} U^i = (I - U)^{-1}. \tag{2.20}$$

Thus, $N_{ij}$ is the expected number of visits to $(k, j)$ before reaching any state of level $k-1$, given that the process starts in $(k, i)$. We subsequently have that

$$R_{hj} = \sum_{1 \leq i \leq n} (A_1)_{hi} N_{ij} \tag{2.21}$$

which can be written in matrix form as

$$R = A_1 N = A_1 (I - U)^{-1}. \tag{2.22}$$

We have then established a link between $R$ and $U$. We can proceed similarly to associate $U$ with the the last of the three matrices we mentioned earlier, that is $G$. We define $G_{ij}$ as, for $k > 0$, the probability of return in $(k, j)$ in a finite time, given that the process starts in $(k+1, i)$. According to a reasoning similar to the previous ones, we can establish that

$$U_{ij} = (A_0)_{ij} + \sum_{1 \leq h \leq n} (A_1)_{ih} G_{hj} \tag{2.23}$$

which can be written as

$$U = A_0 + A_1 G. \tag{2.24}$$

Because the probabilistic interpretation of this relation is very similar to the one we gave earlier about the relation between $U$ and $R$, we do not give it.

From the above relations, we can derive the following theorem, which establishes the thorough relationship between $R$, $U$ and $G$ [18, Section 6.2].

**Theorem 2.1.6 (Relations between the matrices *R*, *G*, and *U*)** *If either R, U or G is determined, then the other two can be computed by applying the following equations:*

$$\begin{aligned} U &= A_0 + A_1 G, && (2.25a)\\ U &= A_0 + R A_{-1}, && (2.25b)\\ R &= A_1 (I - U)^{-1}, && (2.25c)\\ R &= A_1 (I - A_0 - A_1 G)^{-1}, && (2.25d)\\ G &= (I - U)^{-1} A_{-1}, && (2.25e)\\ G &= (I - A_0 - R A_{-1})^{-1} A_{-1}. && (2.25f) \end{aligned}$$

We have not properly introduced the last three equations. However, they can easily be derived from the other three. This theorem is essential, because it implies that in order to compute the stationary vector of an inhomogeneous and infinite QBD, all we have to do is to determine one of the three above matrices. Subsequently, we give next an implementation of the so-called *Logarithmic Reduction* algorithm to compute matrix $G$.

## Development of the algorithm

As we have observed, determining the stationary vector of an aperiodic, irreductible, positive recurrent, discrete time QBD that is homogeneous and infinite comes to computing the rate matrix $R$. We have also showed that a strong relation holds between $R$ and two other matrices, denoted by $U$ and $G$. In particular, once one of the three matrix is determined, the other two can easily be computed. In this section, we present an algorithm that computes an approximation of $G$. Like Latouche and Ramaswami [18, Chapter 8], we choose to focus on matrix $G$ because it has the property of being a stochastic matrix when the QBD is positive recurrent, which permits to ease the definition of an approximation error.

Latouche and Ramaswami [18, Section 8.4] present three algorithms to compute matrix $G$. We are interested only in the most efficient one, called the *Logarithmic Reduction* algorithm. It relies on the following theorem.

**Theorem 2.1.7 (Computation of matrix $G$)**
*Matrix G can be computed as*

$$G = \lim_{k\to\infty} G^{[k]} \tag{2.26}$$

*where, for $k \geq 0$*

$$\begin{aligned}
G^{[k]} &= \textstyle\sum_{k\geq 0}(\prod_{0\leq i\leq k-1} H^{[i]})L^{[k]},\\
L^{[0]} &= (I - A_0)^{-1}A_{-1},\\
H^{[0]} &= (I - A_0)^{-1}A_1,\\
L^{[k+1]} &= (I - U^{[k]})^{-1}(L^{[k]})^2,\\
H^{[k+1]} &= (I - U^{[k]})^{-1}(H^{[k]})^2,\\
U^{[k]} &= H^{[k]}L^{[k]} + L^{[k]}H^{[k]}.
\end{aligned}$$

This theorem establishes that G can be obtained by computing the sum given in Equation (2.26). However, because this sum is infinite, determining it exactly would require an infinite number of computations. Therefore, we develop an algorithm that gives only an approximation of matrix $G$. Because $G$ is a stochastic matrix, the sum of any of its lines must be equal to 1. Thus, we can define the approximation error $\epsilon$ as

$$\epsilon = \mathbf{1}^T|\mathbf{1} - G\mathbf{1}| \tag{2.27}$$

where $M^T$ denotes the transpose of matrix $M$.

An OCTAVE function that computes an approximation of $G$ is shown in Figure 2.2. Its parameters are the matrices $A_{-1}, A_0, A_1$, and $\epsilon$, respectively. Initially, L is equal to $L^{[0]}$, H is equal to $H^{[0]}$, T is the identity matrix and G is equal to L. Then, the loop 8-14 successively computes matrices $G^{[k]}$. More precisely, after the $m^{\text{th}}$ iteration, L is equal to $L^{[m]}$, H is equal to $H^{[m]}$, T is equal to $\prod_{0\leq i\leq m-1} H^{[i]}$ and G is equal to $G^{[k]}$.

Latouche and Ramaswami [18, Section 8.4] give an upper bound of the number of iterations performed by the algorithm for a given error $\epsilon$, denoted by $K_{LR}$,

$$K_{LR} \leq \log_2 \frac{|\log \epsilon|}{|\log \eta|} \tag{2.28}$$

where $\eta$ denotes the spectral radius of matrix $R$. During an iteration, a performed operation is either a matrix addition, matrix multiplication[2] and

[2]We assume that matrix multiplication is performed with the naive algorithm. That is,

```
function G = QBD_LR(A_1, A0, A1, e)
        n = size(G,1)
        I = eye(size(A0)); # identity matrix
        L = (I - A0)^(-1) * A_1;
        H = (I - A0)^(-1) * A1;
        G = L;
        T = I;
        while(1 - sum(G)/n <= e) # computes G(k)
                T = T * H;
                U = H * L + L * H;
                L = (I - U)^(-1) * L^2;
                H = (I - U)^(-1) * H^2;
                G = G + TL;
        end
endfunction
```

Figure 2.2: An OCTAVE function that implements the *Logarithmic Reduction* algorithm.

matrix inversion[3]. Because all the matrices are $n \times n$ sized, where $n$ is the number of phases in any level other than level 0, the worst-case time complexity of these operations are, respectively, $\mathcal{O}(n^2)$, $\mathcal{O}(n^3)$ and $\mathcal{O}(n^3)$ in the number of scalar operations.

**Proposition 2.1.2**
*The worst-case time complexity of the Logarithmic Reduction algorithm is $\mathcal{O}(n^3 K_{LR})$ in the number of scalar operations.*

## 2.1.4 Linear Level Reduction algorithm

In this subsection, we aim to develop an approach to compute the stationary vector of the other types of QBDs. First, we present the approach applied to finite and homogeneous QBDs. Then, we extend it to handle the inhomogeneous case further in the subsection.

Latouche and Ramaswami [18, Section 10.1] establish expressions used to compute $\pi$, which we describe in the following theorem.

each $(AB)_{ij}$ is obtained by computing $\sum_k A(i,k)B(k,j)$. This algorithm has a worst-case time complexity of $\mathcal{O}(n^3)$.

[3]We suppose that matrix inversion is performed by Gauss-Jorgan elimination, which has a worst-case time complexity of $\mathcal{O}(n^3)$.

**Theorem 2.1.8 (Determination of $\pi$ (finite and homogeneous))** *For a homogeneous, finite, irreductible and positive recurrent QBD with maximum level M, the stationary vector $\pi$ is given by*

$$\begin{aligned} \pi_0 &= \pi_0 U^{(0)}, && (2.29) \\ \pi_k &= \pi_{k-1} R^{(k)}, \quad 1 \le k \le M, && (2.30) \end{aligned}$$

*where*

$$\begin{aligned} R^{(1)} &= B_1(I - U^{(1)})^{-1}, && (2.31a) \\ R^{(k)} &= A_1(I - U^{(k)})^{-1}, \quad 2 \le k \le M-1, && (2.31b) \\ R^{(M)} &= C_1(I - U^{(M)})^{-1}, && (2.31c) \\ U^{(M)} &= C_0, && (2.31d) \\ U^{(M-1)} &= A_0 + R^M C_{-1}, && (2.31e) \\ U^{(k)} &= A_0 + R^{(k+1)} A_{-1}, \quad 1 \le k \le M-2, && (2.31f) \\ U^{(0)} &= B_0 + R^{(1)} B_{-1}. && (2.31g) \end{aligned}$$

The similarity between Equations (2.31) and Equations (2.25a) is noteworthy. We can give a similar probabilistic interpretation to both systems of equations. Indeed, for $1 \le k \le M$, matrix $U^{(k)}$ records the probability of return to level $k$ before a visit to level $k-1$, given that the process starts in level $k$. Furthermore, $R^{(k)}$ plays the same role as the rate matrix we defined previously. However, unlike the rate matrix, $R^{(k)}$ depends on the level because the QBD is finite. Therefore, the approach we presented previously cannot be applied.

From Equations (2.31), we develop an algorithm to compute $\pi$, which is implemented in the Octave function shown in Figure 2.3. The parameters of the function are the nine inner blocks that compose the transition matrix, in the following order: $B_{-1}, B_0, B_1, A_{-1}, A_0, A_1, C_{-1}, C_0, C_1$ (see Equation (1.26)). Matrices $U^{(k)}, 0 \le k \le M$, and matrices $R^{(k)}, 1 \le k \le M$, are first computed (lines 2-13). Next, $\pi_0$ is computed by the GLU function (line 16), whose implementation is given in Figure 2.1. Vectors $\pi_k, 1 \le k \le M$, are then computed (lines 19-22). Finally, the whole vector $\pi$ is normalized (lines 25-27).

```
function pi = QBD_LLR(B_1, B0, B1, A_1, A0, A1, C_1, C0, C1, M)
        # Computes matrices R(i) and U(i)
        I = eye(size(C0));
        U{M+1} = C0;
        R{M} = C1 * (I- U{M+1})^(-1);
        I = eye(size(A0));
        U{M} = A0 + R{M} C_1;
        for i = M-2:-1:1
                R{i+1} = A1 * (I- U{i+2})^(-1);
                U{i+1} = A0 + R{i+1} * A_1;
        end
        R{1} = B1 * (I- U{1})^(-1);
        U{0+1} = B0 + R{1} * B_1;

        # Computes the solution of pi(0) = pi(0) U(0)
        pi{0+1} = GLU(U{0+1});
        # Computes pi(i)
        sump = sum(pi{0+1});
        for i = 1:M
                pi{i+1} = pi{i} * R{i};
                sump = sump + sum(pi{i+1});
        end

        # Normalizes
        for i = 1:M+1
                pi{i} = pi{i} / sump;
        end
endfunction
```

Figure 2.3: An Octave function that computes the stationary vector of a homogeneous and finite QBD.

**Proposition 2.1.3**
*The worst-case time complexity of the Linear Level Reduction algorithm is $\mathcal{O}(Mn^3)$ in the number of scalar operations, where $n$ is the number of rows and columns of matrix $A_0$.*

More efficient algorithms are presented in Latouche and Ramaswami [18, Chapter 10]. However, as we will see in Section 2.1.4, the one shown in Figure 2.3 can easily be generalized in order to be applied to inhomogeneous QBDs. This is the very reason why we chose to present it.

### Linear Level Reduction algorithm for inhomogeneous QBDs

In this section, we finally discuss how to compute the stationary vector of an inhomogeneous QBD. The algorithm we present is merely a generalization of the one presented in Figure 2.3. Because the QBD is inhomogeneous, the computation of $\pi$ requires to determine several rate matrices, that is one per level. The key to compute these matrices is a system of equations similar to Equations (2.31). However, this system of equations is only intended for finite QBDs. We suppose for now that the QBD is finite and we show how to handle the infinite case a little bit further in this section. The following theorem, established by Latouche and Ramaswami [18, Section 6.2], is a direct generalization of Theorem 2.1.8.

**Theorem 2.1.9 (Determination of $\boldsymbol{\pi}$ (inhomogeneous))**
*The stationary vector of an inhomogeneous, finite, aperiodic, irreductible, and positive recurrent QBD is given by*

$$\pi_0 = \pi_0 U^{(0)} \tag{2.32}$$
$$\pi_k = \pi_{k-1} R^{(k)}, \quad k \geq 1 \tag{2.33}$$

*for* $1 \leq k \leq M-1$, *where*

$$R^{(k)} = A_1^{(k)}(I - U^{(k)})^{-1} \tag{2.34a}$$
$$U^{(M)} = A_0^{(M)} \tag{2.34b}$$
$$U^{(k)} = A_0^{(k)} + R^{(k+1)} A_{-1}^{(k+1)} \tag{2.34c}$$

When the inhomogeneous QBD is infinite, Latouche and Ramaswami [18, 12.2] advise to choose an arbitrary but large level $M$ and to truncate the process at level $M$. We expect $M$ to be such that the truncation does not alter vector $\pi$ too much, that is $M$ must be sufficiently large for the stationary probability of levels higher than $M$ to be negligible. The selection of a good value of $M$ is generally specific to the considered QBD. Another essential question is how to gather the three inner blocks $A_{-1}^{(M)}, A_0^{(M)}$, and $A_1^{(M)}$ into two blocks $A_{-1}^{'(M)}$ and $A_0^{'(M)}$ of the truncated process in order to be able to compute matrix $U^{(M)}$, as it is defined in Equations (2.34). However, we do not deal with these two issues in this thesis.

Finally, we present in Figure 2.4 an Octave function implementing the algorithm for computing $\pi$ when the QBD is inhomogeneous. As stated above, this algorithm clearly appears as a generalization of the one illustrated in Figure 2.3. Only the first part of the new algorithm (lines 3-11) is different from the previous one. A noticeable difference is the dynamic computation of the inner blocks composing the transition matrix. Indeed, they are computed by the function `getInnerBlocks`, which requires only one parameter, namely

```
 1 function pi = QBD_LLR_INHOM(M)
 2         # Computes matrices R(i) and U(i)
 3         [A_1 A0 A1] = getInnerBlocks(M)
 4         U{M+1} = A0;
 5         for i = M-1:-1:0
 6                 I = eye(size(A0));
 7                 Upper-A_1 = A_1;
 8                 [A_1 A_0 A1] = getInnerBlocks(i)
 9                 R{i+1} = A1 * (I- U{i+2})^(-1);
10                 U{i+1} = A0 + R{i+1} * A_1;
11         end
12
13         # Computes the solution of pi(0) = pi(0) U(0)
14         pi{0+1} = GLU(U{0+1});
15         # Computes pi(i)
16         sump = sum(pi{0+1});
17         for i = 1:M
18                 pi{i+1} = pi{i} * R{i};
19                 sump += sum(pi{i+1});
20         end
21
22         # Normalizes
23         for i = 1:M+1
24                 pi{i} = pi{i} / sump;
25         end
26 endfunction
```

Figure 2.4: An Octave function that computes the stationary vector of an inhomogeneous and finite QBD.

the level to compute the blocks of. More precisely, at the end of the kth iteration of the first loop (lines 4 - 10), `A_1` is $A_{-1}^{(M-1-k)}$,[4] `A0` is $A_0^{(M-1-k)}$, `A1` is $A_1^{(M-1-k)}$, and `Upper-A_1` is $A_{-1}^{(M-k)}$. The time complexity of the algorithm is $\mathcal{O}(n^3)$ in the number of scalar operations, where $n$ is defined as the number of rows of the largest inner block $A_0^{(i)}$. However, we must be aware that in practice, the run time of the function is very sensitive to how the function `getInnerBlocks` is implemented.

## 2.2 First passage times

The previous section is dedicated to the definition of the stationary distribution as a relevant performance measure, and to the development of algorithms that compute it. In this section, we present another measure essential for the performance: the expected first passage times. We immediately con-

[4]Note that if k = M-1, matrix $A_{-1}^{(M-1-k)}$ does not exist. By convention, we replace it with an empty matrix

sider the inhomogeneous QBD and we arbitrarily assume that the QBD is infinite. Because considering the homogeneous and/or finite QBDs would require only small adaptations, we do not explicitly give the related algorithms.

Intuitively, the first passage times describe the time needed by the process to reach a given state (or a given set of states), starting from another given state (or another given set of states). More formally, we define them as follows.

**Definition 2.2.1 (Expected first passage time)**
*Let $\tau(k)$ be the first passage time to level $k$, that is*

$$\tau(k) = inf\{t > 0 : X(t) \in l(k)\} \tag{2.35}$$

*Then, we denote by $t_{(i,j)\to k}$ the expected first passage time from state $(i, j)$ to level $k$, which we define as*

$$t_{(i,j)\to k} = E[\tau(k)|X(0) = (i, j)] \tag{2.36}$$

*Finally, we define the expected first passage time from level $i$ to level $k$, written $t_{i\to k}$ as*

$$\begin{aligned} t_{i\to k} &= E[\tau(k)|X(0) \in l(i))] \\ &= \sum_j E[\tau(k)|X(0) = (i, j)] \frac{\pi_i(j)}{\sum_l \pi_i(l)} \end{aligned} \tag{2.37}$$

In the following example, we illustrate the usefulness of the expected first passage times in performance analysis.

**Example 2.2.1** *Let us consider an M/PH/1 queueing system similar to the one defined in Example 1.3.1, except that we limit the queue capacity to 10 requests. We model it with a QBD similar to the one in Example 1.3.1, except that the number of levels is 11. We want to know the expected time needed by the empty system to be at full capacity, that is for the QBD to reach level 10, starting in level 0. This time is expected first passage time from level 0 to level 10.*

Next, we denote by $\Gamma_{i\to k}$ the first passage probabilities from level $i$ to level $k$, namely

$$(\Gamma_{i\to k})_{jl} = P[X(\tau(k)) = l|X(0) = (i, j)] \tag{2.38}$$

that is, $(\Gamma_{i\to k})_{jl}$ gives the probability that the state visited by the process the very first time the process reaches level $k$ is $(k, l)$, given that it starts from $(i, j)$. It is noteworthy that we have, for $i \geq 0$,

$$t_{i\to i} = \mathbf{0}^T \tag{2.39}$$

$$\Gamma_{i\to i} = I \tag{2.40}$$

where $\mathbf{0}$ is a row vector of appropriate size and filled with zeros.

Latouche and Ramaswami [18, Section 11.2] present a method to compute the expected first passage times from level 0 to any upper level of a homogeneous QBD. We extend their approach to the computation of the expected first passage times from any level to any other level and we apply this extension to the inhomogeneous case. In order to ease the reading, we first show how to determine $t_{i\to k}$ when $i < k$, then we give the instructions to adapt our algorithm to the case where $i > k$.

### 2.2.1 Computing first passage times upward

Our approach makes intensively use of the restrictions imposed on the transitions of a QBD. Because the process can move only within a one-level range, reaching level $k$ from level $i$, with $i < k$, requires the process to go through levels $i+1, i+2, \ldots, k$. The first passage time from state $(i, j)$ to level $k$ is therefore equal to the time to reach level $i+1$ plus the time to reach level $k$, starting from a state of level $i+1$ balanced with regard to its first passage probabilities from $(i, j)$. More formally, for $0 \leq i < k$, we have

$$t_{i\to k} = t_{i\to i+1} + \Gamma_{i\to i+1} t_{i+1\to k}. \tag{2.41}$$

Similarly, $\Gamma_{i\to k}$, can be recursively computed, because going from level $i$ to level $k$ requires to go through levels $i+1, i+2, ..., k$. Then, we have that

$$\Gamma_{i\to k} = \Gamma_{i\to i+1} \Gamma_{i+1\to k}. \tag{2.42}$$

If we successively apply Equation (2.41) to compute $t_{i+1\to k}, t_{i+2\to k}, ..., t_{k-1\to k}$, we obtain

$$t_{i\to k} = \sum_{i\leq j<k} \Gamma_{i\to j} t_{j\to j+1} \tag{2.43}$$

where, as a result of Equation (2.42),

$$\Gamma_{i\to k} = \prod_{i\leq j<k} \Gamma_{j\to j+1}. \tag{2.44}$$

To compute $t_{i\to k}$ and $\Gamma_{i\to k}$, we must therefore determine some expressions to compute, for $i \geq 0$, $t_{i\to i+1}$ and $\Gamma_{i\to i+1}$. These expressions are given in the following theorem. We refer to Latouche and Ramaswami [18, Section 11.2] for a formal proof.

**Theorem 2.2.1 (Computation of upward first passage times)**
*For $i \geq 0$, we have that*

$$\Gamma_{i\to i+1} = (I - D_i)^{-1} A_1^{(i)} \tag{2.45a}$$
$$t_{0\to 1} = (I - D_0)^{-1} \boldsymbol{1} \tag{2.45b}$$
$$t_{i\to i+1} = (I - D_i)^{-1} \boldsymbol{1} + (I - D_i)^{-1} A_{-1} t_{i-1\to i} \tag{2.45c}$$

*where the matrices $D_i$ are defined as*

$$D_0 = B_0 \tag{2.46a}$$
$$\begin{aligned} D_i &= A_0^{(i)} + A_{-1}^{(i)} (I - D_{i-1})^{-1} A_1^{(i-1)} \\ &= A_0^{(i)} + A_{-1}^{(i)} \Gamma_{i-1\to i} \end{aligned} \tag{2.46b}$$

Matrix $D_i$ records the probability of return to level $i$ before a visit to level $i+1$, given that the process starts in level $i$. To determine it, we need to consider that the process can either

1. move directly from a state of level $i$ to another state of this same level,

2. or it can go down a level before it eventually reaches back level $i$,

hence Equation (2.46). Equation (2.45c) expresses that in order to determine $t_{i\to i+1}$, we need to consider not only the expected time spent in level $i$ before reaching level $i+1$, but also the possibility to move down to level $i-1$ and therefore, the expected time that would be needed to go back up in level $i$.

Thanks to the above equations, we develop an algorithm to compute the first passage times from a given level to any upper level of an inhomogeneous QBD. We implement this algorithm in the OCTAVE function shown in Figure 2.5. Its parameters are, respectively, the starting level, denoted by $S$, and the target level, denoted by $M$. First, $C_0$, $T_{0\to 1}$, and $t_{0\to 1}$ are computed (lines 2-7). Then the first loop is executed (lines 10-16). At the end of the ith iteration, for $1 \leq i \leq M-1$, we compute $C_i$, $T_{i\to i+1}$, and $t_{i\to i+1}$. Finally, the second loop (lines 19-21) is executed. The jth iteration determines $t_{M-2-j\to M}$, for $1 \leq j \leq M-2-S$.

```
function time = QBD_fpt_up(S, M)
        # Computes the fpp and the fpt from level 0 to level 1
        [A_1 A0 A1] = getInnerBlocks(0);
        D{0+1} = A0;
        Inv = (eye(size(D{0+1})) - D{0+1})^-1;
        T{1} = Inv * A1;
        t{1} = sum(Inv')';
        # Computes the fpp and the fpt from level i to level i+1,
        # 1 <= i <= M
        for i = 1:M-1
                [A_1 A0 A1] = getInnerBlocks(i);
                D{i+1} = A0 + A_1 * T{i};
                Inv = (eye(size(D{i+1})) - D{i+1})^-1;
                T{i+1} = Inv * A1;
                t{i+1} = Inv * (ones(size(Inv,2),1) + A_1 * t{i});
        end
        # Computes the fpt from level i to level M, S <= i <= M-1
        fpt{M} = t{M};
        for i = M-2:-1:S
                fpt{i+1} = t{i+1} + T{i+1} * fpt{i+2};
        end

        time = fpt{S+1};
endfunction
```

Figure 2.5: An Octave function that computes the first passage time from level $S$ to level $M$ of an inhomogeneous QBD.

**Proposition 2.2.1**
*The worst-case time complexity of the algorithm that computes the first passage times upward is $\mathcal{O}(n^3 M)$ in the number of scalar operations, where $n$ is the number of rows and columns of the largest square inner block $A_0(i), i \geq 0$, and $M$ is the target level.*

Indeed, the most complex part of the algorithm executes $M-1$ iterations of a loop that performs a few $n^3$ operations.

### 2.2.2 Computing first passage times downward

In this subsection, we adapt the above approach to compute the expected first passage times from a given level to a lower one. First, in case of infinite QBD, we must truncate the process to an arbitrary but large level, denoted by M. In the finite case, $M$ is equal to the maximum level of the process. We adapt the previously defined relations in order to determine the expected first passage times and probabilities from level $i$ to level $k$, $i > k$. We have, for $0 \leq k < i \leq M$, that

$$\begin{aligned}
\Gamma_{i\to i-1} &= (I - D_i)^{-1} A_{-1}^{(i)} && \text{(2.47a)}\\
\Gamma_{i\to k} &= \Gamma_{i\to i-1}\Gamma_{i-1\to k} && \text{(2.47b)}\\
\mathbf{t}_{M\to M-1} &= (I - D_M)^{-1}\mathbf{1} && \text{(2.47c)}\\
\mathbf{t}_{i-1\to i-2} &= (I - D_{i-1})^{-1}\mathbf{1} + (I - D_{i-1})^{-1} A_1^{i-1}\mathbf{t}_{i\to i-1} && \text{(2.47d)}\\
\mathbf{t}_{i\to k} &= \mathbf{t}_{i\to i-1} + \Gamma_{i\to i-1}\mathbf{t}_{i-1\to k} && \text{(2.47e)}
\end{aligned}$$

where matrix $D_k$, $1 \leq k \leq M$, records the probability of return to level $k$ before reaching level $k-1$, given that the process starts in level $k$. When the QBD has a finite number of levels, we have, for $0 < k < M$, that

$$\begin{aligned}
D_M &= C_0, && \text{(2.48a)}\\
D_k &= A_0^{(k)} + A_1^{(k)}(I - D_{k+1})^{-1}A_{-1}^{(k)} \\
&= A_0^{(k)} + A_1^{(k)}\Gamma_{k+1\to k}. && \text{(2.48b)}
\end{aligned}$$

If the QBD is infinite, $D_M$ must be approximated. As we previously mentioned, we do not discuss how to get a suitable approximation. Thanks to Equations (2.47), we can develop an algorithm that computes the expected first passage times from any level to any lower level. We do not explicitly give it, because it is similar to the one presented in Figure 2.5.

## 2.3 Conclusion

In this chapter, we introduced the matrix analytic methods. Thanks to these ones, we developed algorithms that either determine explicitly the stationary vector and the first passage times of a discrete time QBD, or compute a good approximation of them.

By using the uniformization technique, we can also extend our methods to the continuous time QBD. However, when the content of the inner blocks composing the QBD is complex, the run time of the algorithms can be significant. In order to overcome this relative lack of efficiency, we present in the next chapter some simulation approaches that can be used to determine an estimate of the performance measures more rapidly.

# Chapter 3

# Discrete-Event Simulation

In the previous chapter, we introduced the matrix analytic methods and presented the concept of steady state. We also defined two measures that are relevant for the performance of the studied system, namely the stationary probability vector and the expected first passage times. Finally, we developed algorithms that compute these measures. These algorithms are based on the matrix analytic methods. We also showed that they all have a polynomial time complexity. However, the run time of a program executing these algorithms can be significant, especially when the inner blocks composing the transition matrix or the infinitesimal generator of the considered QBD are huge matrices with a complex structure. To overcome this drawback, we discuss the simulation approaches in this chapter and we build a simulation algorithm for QBDs according to the principles of these approaches. Basically, instead of computing the desired measures, our algorithm is used to simulate the behaviour of the studied process. Thanks to the simulation results, we can estimate the value of these measures. The simulation algorithm is part of our tool, that we describe in Chapter 4.

First, we present a taxonomy of the simulation models in Section 3.1. Then, we focus on a particular type of simulation model, called the *discrete-event simulation model*. We especially put emphasis on an approach to discrete-event simulation, namely the *next-event simulation*. In Section 3.2, we apply the principles of the next-event simulation to the development of an algorithm that simulates a Quasi-Birth-Death process. Simulating a stochastic process requires the generation of random numbers. Subsequently, Section 3.3 briefly presents methods to produce random numbers sampled from either a discrete probability distribution or a continuous distribution. Another critical step of a simulation-based analysis is the data estimation. Hence, in Section 3.4, we discuss how to estimate the performance measures from the simulation results.

## 3.1 Next-event simulation

In order to simulate a Quasi-Birth-and-Death process, we use a specific simulation approach called *Next-event*. Leemis and Park [19, Chapter 5] present the next-event simulation applied to a specific type of model, *i.e.* a discrete-event simulation model. In this section, we discuss first the taxonomy of simulation models given by Leemis and Park [19, Section 1.1]. The taxonomy distinguishes whether a simulation model is:

**Deterministic or stochastic** – The outcome of a deterministic model is not subject to random variations. In other words, two identically parameterized simulations of a deterministic model give the same results. On the other hand, a stochastic model has at least one random component.

**Static or dynamic** – A simulation model is called dynamic if time has a significant influence on the system behaviour. Otherwise, it is called static.

**Continuous or discrete** – A continuous simulation model has state variables that evolve continuously over time. If the state variables evolve only at discrete times, the simulation model is called discrete.

According to this taxonomy, Leemis and Park [19, Section 1.1] define a discrete-event simulation model as follows.

**Definition 3.1.1 (Discrete-event simulation model)**
*A* ***discrete-event simulation model*** *is a simulation model with the following features:*

- *stochastic: at least one of the system variables is random;*
- *dynamic: time has a significant influence on the evolution of the system variables;*
- *discrete-event: the key changes in the system state are due to events that occur at discrete times.*

It is worth mentioning that in this definition, the word *event* refers to an occurrence that can change the state of the considered system, unlike the definition we gave at the beginning of Chapter 1. Obviously, a QBD can be simulated by a discrete-event simulation model. Indeed, it is a stochastic process, which is commonly used to model the evolution of random systems over time. Furthermore, the state modifications in such a process occur at discrete times. This last property obviously holds for discrete time QBDs. It also holds for continuous time QBDs as well. Indeed, we know that the state of a continuous time QBD can only change at epochs determined by an exponential distribution.

For now, we consider only discrete time processes. We specifically develop simulation algorithms for this type of QBD. At the end of the chapter, we discuss the modifications needed for the algorithms to be applied to continuous time QBDs.

Next, we explain the principles of a discrete-event simulation approach called *next-event*. A discrete-event simulation model build via the next-event approach includes several components [19, Section 5.1].

1. *State variables*: At any time, the state of the simulated system is given by the value of some variables called the state variables.

2. *Events*: The occurrences that can modify the value of the state variables are called events.

3. *Event types*: Each event is associated with an event type, which determines how the considered event can influence the system state.

4. *Simulation clock*: Because a discrete-event simulation model is dynamic, we must keep track of the simulation time. Hence, we call "simulation clock" the data structure that records the time elapsed since the beginning of the simulation.

5. *Scheduler*: Because several instances of distinct event types can occur, it is necessary to use a data structure that ensure the correct scheduling of these instances. This data structure is called scheduler.

After defining these components, Leemis and Park [19, Section 5.1] give the four main steps of a next-event simulation.

1. *Initialise*: The simulation clock time is set to zero. The first occurrences are placed in the scheduler.

2. *Process the next event*: The next event to occur is obtained from the scheduler. The simulation clock time is set to the occurrence time of the event and the state variables are updated with respect to the features of the event.

3. *Schedule new events*: The occurrence of an event can generate new events to occur. These ones are placed in the scheduler.

4. *Terminate*: Once some terminal conditions are met, the simulation ends and the output analysis begins. For example, the simulation can end when a special event occurs, when a given number of events have occurred, when the simulation clock reaches a given time, or when the estimate of a given measure reaches a defined precision.

In the following subsection, we adapt the next-event approach in order to simulate a QBD process.

## 3.2 Simulating a Quasi-Birth-Death process

While our goal remains the simulation of a QBD in order to estimate its stationary distribution and the expected first passage times, we present first the simulation of a general Markov process based on the principles of the next-event approach. This first algorithm serves as a basis for the development of the actual QBD-simulating algorithm. For now, we consider a discrete time Markov process. For recall, such a process is defined by a set of random variables indexed with a parameter:

$$\mathcal{M} = \{X(t), t \in \mathcal{T}\} \tag{3.1}$$

where $\mathcal{T}$ is a countable set. In our simulation model, we define $\mathcal{T}$ as $\mathbb{N}$ and consider that each $t$ models a time unit. In other words, $X(t)$ is the state of the process at the $t^{\text{th}}$ time unit. Our simulation algorithm makes essentially use of the Markov property. For recall, if this property holds, then the probability of reaching a given state at the $(t+1)^{\text{th}}$ time unit can be determined knowing only the state of the process at the $t^{\text{th}}$ time unit and the transition probabilities associated with this state.

According to the principles of next-event simulation detailed in Subsection 3.1, we define the simulation model as follows.

1. We only need one variable to model the state of the process, which merely records the current state of the process itself.

2. The only events that can occur during the simulation are the possible transitions of the process.

3. The only event types is the firing of a transition.

4. Because we consider the discrete time Markov process, we define a time unit as the time between two transitions. In other words, we model the simulation clock with an integer, which is incremented each time a transition is fired.

```
function result = sim_Markov(T, initialState, endTime)
        state = initialState;
        clock = 0;
        while(clock < endTime)
                output = setOutput(state, clock, output);
                transProbs = T(state,1:end);
                state = getNextState(transProbs);
                clock++;
        end
        result = estimate(output);
endfunction
```

Figure 3.1: An OCTAVE function that simulates a Markov process.

5. The scheduler is not explicitly required because one and only one transition is fired at each time unit. At any time, the scheduler would therefore contain only the next transition to occur.

Finally, we choose to stop the simulation once a given number of transitions have been fired. However, as we discuss later in this chapter, it requires cautiousness with regard to the produced estimates of the performance measures. In particular, we are aware that at the end of a simulation, the steady state of the simulated process may have not been reached.

Accordingly, we give in Figure 3.1 an OCTAVE function that simulates a discrete time Markov process. The function has three parameters: `T` is the transition matrix associated with the process; `initialState` is the state in which the process starts; and `endTime` is the simulation time at which the simulation ends. It is noteworthy that although the initial state of the process can be chosen, it is usually set as zero. First, the simulation clock is initialised and the current state is set to `initialState`. Next, we enter the simulation loop. We register the current state and clock time in a data structure denoted `output` by calling the function `setOutput`. This data structure is needed for further output analysis. The variable `transProbs` is a vector recording the transition probabilities from the current state $i$, that is

$$\forall j \in \mathcal{S} : \texttt{transProbs}(j) = \texttt{T}(i, j)$$

The next state of the process is determined according to the transition probabilities recorded in `transProbs` and the simulation time is incremented. The loop is repeated while the clock time has not reached the end of the simulation. Finally, the function `estimate` estimates the performance measure thanks to the content of the data structure `output`.

For now, we avoid explaining how to deal with the randomness and the output production. These issues are discussed in Section 3.3 and Section 3.4, respectively. Instead, we adapt the algorithm in order to take into account the specific structure of a QBD. More precisely, we assume that the QBD is inhomogeneous and we discuss the homogeneous case further in this section.

An Octave function that implements the new algorithm is presented in Figure 3.2. First, because a QBD has a two-dimensional state space, two state variables are required: `level` records the current level of the process and `phase` records its current phase. Subsequently, our algorithm requires two parameters to define the initial state: `initialLevel` and `initialPhase`, which are respectively the initial level and the initial phase. The signature of the function `setOutput` is also modified accordingly.

In the function shown in Figure 3.1, `transProbs` recorded all the transition probabilities from the current state. When the process is a QBD, most of its elements are zeros. Therefore, if we denote the current state of the QBD by $(k, j)$, we can consider only the $j^{\text{th}}$ row of matrices $A_{-1}^{(k)}$, $A_0^{(k)}$, and $A_1^{(k)}$. For this purpose, we define the function `getInnerBlocks`, which returns the three inner blocks of a level given as parameter. It is noteworthy that we already needed such a function in Chapter 2. As already stated in the previous chapter, calling this function can be time consuming. To avoid unnecessary calls to this function we apply the following optimization: when a level is reached for the first time, we record the related inner blocks for further use in a data structure called `container`. For the sake of readability, we assume that this feature is implemented in `getInnerBlocks`. Therefore, this function requires a second parameter, *i.e.* `container`. The function possibly modifies `container` and returns it in addition to the three inner blocks.

In order to determine the next state, we need to know the current level, the current phase, and the three inner blocks related to the current level. The signature of the function `getNextState` is modified accordingly and it now requires these five parameters.

We still have to give the implementation of the intermediary functions `setOutput`, `getInnerBlocks`, `getNextState`, and `estimate`. We have already given the outlines of `getInnerBlocks` and its implementation depends on the studied QBD. We deal with functions `setOutput` and `estimate` in Section 3.4, where we present some estimation methods. The random number generation needed by `getNextState` is presented in Section 3.3.

It is noteworthy that the function presented in Figure 3.2 can also be used when the QBD is homogeneous. In this case, the function `getInnerBlocks` returns always the same matrices, except when the process is in a boundary

```
function result = sim_QBD(initialLevel, initialPhase, endTime)
        level = initialLevel;
        phase = initialPhase;
        clock = 0;
        container = [];
        while(clock < endTime)
                output = setOutput(level, phase, clock, output);
                [A_1 A0 A1 container] =
                        getInnerBlocks(level, container);
                [level phase] =
                        getNextState(level, phase, A_1, A0, A1);
                clock++;
        end
        result = estimate(output);
endfunction
```

Figure 3.2: An OCTAVE function to simulate a Quasi-Birth-and-Death process.

level[1]. Therefore, we can avoid a function call by modifying the signature of the algorithm. After the modification, the algorithm would require the different matrices composing the transition matrix as parameters.

## 3.3 Random number generator

In this section, we present algorithms to compute random numbers. Such an algorithm is called **random number generators** (RNG). It is worth mentioning that most of the programming languages have native implementations of random number generators. However, they do not always have all the properties we require, hence the purpose of this section.

First, we present in Subsection 3.3.1 a RNG that computes uniformly distributed real numbers in $(0, 1)$. Then, we describe methods to build RNGs that generate numbers following a given discrete probability distribution (Subsection 3.3.2) or a continuous one (Subsection 3.3.3). We stress that this section is only an introduction to the random number generator theory. We refer to Leemis and Park [19, Chapter 2] and Knuth [17, Chapter 3] for a more thorough presentation.

### 3.3.1 Linear congruential generator

Our first goal is to produce real-valued random numbers between 0 and 1. However, there is an infinite number of values in $(0, 1)$. Therefore, the

[1]The boundary levels are level 0 and, if the QBD is finite, the highest level.

algorithms we present generate random numbers from only a finite subset of this interval. The main issue is to develop a *good* generator. Indeed, as mentioned by Knuth [17, Chapter 3], a good generator must have the following properties:

**Uniformity** – Any value in the set has an equal chance to be generated.

**Unpredictability** – Knowing only the last generated value, it is impossible to determine the value that will be generated next.

**Reproducibility** – Any stream of generated random numbers can be reproduced at will.

**Full periodicity** – The sequence of numbers repeated by the generator must be as long as possible.

Leemis and Park [19, Section 2.1] give an algorithm developed by Lehmer that defines a uniform, unpredictable, and reproducible RNG. This algorithm generates random integers. We define it and show how to produce random numbers of a finite subset of $(0, 1)$ instead of a set of integers.

**Definition 3.3.1 (Lehmer's algorithm)**
***Lehmer's algorithm*** *is a random number generation algorithm that is defined by three parameters*

- *modulus $m$, a large integer*
- *multiplier $a$, an integer in $\mathcal{X}_m$*
- *initial seed $x_0 \in \mathcal{X}_m$*

*and the generation function $g : \mathcal{X}_m \longrightarrow \mathcal{X}_m$ such that*

$$g(x) = ax \mod m \tag{3.2}$$

*where*

$$\mathcal{X}_m = \{1, 2, \dots, m-1\}. \tag{3.3}$$

*Therefore, the sequence $x_0, x_1, x_2, \dots$ generated by the algorithm is given by*

$$x_{i+1} = g(x_i), \tag{3.4a}$$
$$x_i \in \mathcal{X}_m. \tag{3.4b}$$

It is important to avoid generating the value zero. Indeed, if the generated number is zero, then every next number is also 0 because of the definition of the generation function. Indeed, we have that

$$g(0) = 0. \tag{3.5}$$

Lehmer's algorithm therefore generates random integers in $(0, m)$. If we divide the produced numbers by the modulus $m$, we obtain real-valued random numbers in $(0, 1)$. Of course, every number in $(0, 1)$ cannot be generated by a single Lehmer's algorithm. More precisely, the higher $m$, the more distinct values can be produced.

Lehmer's algorithm is a particular case of a more general type of RNG called **linear congruential generator** (LCG) [17, Section 3.2]. In addition to the parameters required by Lehmer's algorithm, we define a fourth parameter, called **increment** and denoted by $b$. Then, the generation function $g$ becomes

$$g(x) = (ax + b) \mod m \tag{3.6}$$

We immediately see that Lehmer's algorithm is a linear congruential generator with $b = 0$. It is noteworthy that if $b \neq 0$, then generating the value zero becomes acceptable.

An important property of an LCG is its fundamental period. Because of the definition of the generating function, we have, for every sequence $x_0, x_1, x_2, \ldots$, that

$$\forall i, j \in \mathbb{N} : x_i = x_j \Rightarrow \forall k \in \mathbb{N} : x_{i+k} = x_{j+k}. \tag{3.7}$$

Hence, once a previously generated number is repeated, the whole sequence following it is also repeated afterwards. The unpredictability property is therefore lost. Indeed, every following value is known and cannot be considered as random anymore. Hence, we want to avoid a repetition for a time as long as possible. More formally, we define the fundamental period as follows [19, Section 2.1].

**Definition 3.3.2 (Fundamental period)**
*Let $x_0 \in \mathcal{X}_m$ and $x_0, x_1, \ldots$ the sequence produced by a linear congruential generator with modulus $m$, multiplier $a$, and increment $b$. Then, there exists $p \in \mathbb{N}_0$, with $p \leq m$ such that*

$$\forall i \in \mathbb{N} : x_i = x_{i+p}. \tag{3.8}$$

*We call **fundamental period** of the RNG the smallest integer $p$ satisfying Equation (3.8).*

Because the randomness is essential in simulation, we must avoid reaching the epoch at which the random number sequence is repeated. Therefore, we want the period to be as large as possible. In particular, we define the following special class of LCGs.

**Definition 3.3.3 (Full-period generator)**
*A linear congruential generator with modulus $m$, increment $b$, and fundamental period $p$ has a* ***full period*** *if and only if*

- $b = 0$ *and* $p = m - 1$, *or*
- $b > 0$ *and* $p = m$.

Knuth [17, Section 3.2] gives the conditions for a LCG to have a full period. These conditions are different depending on the value of the increment. First, the following theorem establishes them when $b \neq 0$.

**Theorem 3.3.1 (Full-period conditions ($b \neq 0$))**
*A linear congruential generator with modulus $m$, multiplier $a$, and increment $b \neq 0$ has a full period if and only if it meets the following conditions:*

- *$b$ is relatively prime to $m$, that is they have no common divisor other than 1*
- *for every prime $q$ dividing $m$, $a - 1$ is a multiple of $q$*
- *if $m$ is a multiple of 4, $a - 1$ is a multiple of 4.*

For a Lehmer's generator, *i.e.* a LCG with $b = 0$, the conditions required to have a full period are different. They are given by Knuth [17, Section 3.2] in the following theorem.

**Theorem 3.3.2 (Full-period conditions ($b = 0$))**
*A linear congruential generator with modulus $m$, multiplier $a$, and increment 0 has a full period if and only if it meets the following conditions:*

- *$m$ is prime*
- *initial seed $x_0$ is relatively prime to $m$*
- *$a$ is prime modulo $m$, that is $m - 1$ is the smallest integer $h$ such that $a^h \mod m = 0$*

Thanks to these two theorems, a uniform, unpredictable, reproducible, and full period generator that produces random real-valued numbers in $(0, 1)$ can be built. Next, we present methods to generate random values that follow a given probability distribution. We focus first on discrete probability distributions.

### 3.3.2 Generating discrete random variables

In this section, we discuss how to randomly generate discrete random variables that follow a given probability distribution. First, we define the following function [19, Section 6.1].

**Definition 3.3.4 (Inverse distribution function (discrete))**
*Let $X$ be a discrete random variable, let $\mathcal{S}$ be its state space, and let $F$ be its cumulative distribution function. Then, we define the* ***inverse distribution function (IDF)*** *of $X$ as the function $F^\star : (0, 1) \longrightarrow \mathcal{S}$ such that*

$$\forall u \in (0, 1) : F^\star(u) = \min_{x \in \mathcal{S}} \{x : u < F(x)\} \tag{3.9}$$

Thanks to the definition of IDF, we can produce random numbers following a given probability distribution. Indeed, the following theorem establishes the link between a uniform probability distribution and any discrete probability distribution [19, Section 6.2].

**Theorem 3.3.3 (Probability integral transformation (discrete))** *Let $X$ be a discrete random variable, let $F^\star$ be its IDF, and let $U$ be a continuous random variable following a uniform distribution on $[0,1]$. If we denote by $Z$ the discrete random variable defined by*

$$Z = F^\star(U) \tag{3.10}$$

*then $Z$ and $X$ are identically distributed.*

Using this result, we develop a discrete random variable generator from the IDF of a given random variable and a LCG generating random numbers in $(0,1)$. If we denote by

$$x_0, x_1, \dots, x_n$$

the sequence produced by the LCG, and by $F^\star$ the IDF of a given discrete distribution, then the sequence

$$F^\star(x_0), F^\star(x_1), \dots, F^\star(x_n)$$

follows the considered probability distribution.

Thanks to the above results, we can implement the function `getNextState` we mentioned in Figure 3.2. For recall, its parameters are, respectively, the current level of the process $k$, its current phase $i$, and the three inner blocks related to the current level, namely $A_{-1}^{(k)}, A_0^{(k)}, A_1^{(k)}$. We define the matrix

$$A = \begin{pmatrix} A_{-1}^{(k)} & A_{-1}^{(k)} & A_{-1}^{(k)} \end{pmatrix} \tag{3.11}$$

and we consider the row of $A$ corresponding with the phase $i$. We obtain a probability vector. Each element of this vector gives the probability of transiting to a given state reachable from the current state $(k, i)$. From this vector, we build a CDF-like function. Then, we use a LCG and the IDF related to the CDF we have defined to determine the next state of the QBD.

### 3.3.3 Generating continuous random variables

This section is dedicated to the generation of continuous random variables. The method presented is similar to the one we used to generate discrete random variables. As we did in the previous section, we define the inverse distribution function of a continuous random variable [19, Section 7.2].

**Definition 3.3.5 (Inverse distribution function (continuous))** *Let $X$ be a continuous random variable, let $\mathcal{S}$ be its state space, and let $F$ be its cumulative distribution function. Then, we define the* ***inverse distribution function (IDF)*** *of $X$ as the inverse function of $F$, denoted $F^{-1}$, if it exists.*

By way of illustration, we give the IDF of the exponential distribution in the following example.

**Example 3.3.1** *Let $F$ be the cumulative distribution function of an exponential distribution with rate $\lambda$. Then, the inverse distribution function of the distribution is given by*

$$F^{-1}(u) = -\lambda \ln(1 - u) \tag{3.12}$$

*with* $0 < u < 1$.

Based on this definition, Leemis and Park [19, ] give the continuous version of Theorem 3.3.3.

**Theorem 3.3.4 (Probability integral transformation (continuous))** *Let $X$ be a continuous random variable, let $F^{-1}$ be its IDF, and let $U$ be a continuous random variable following a uniform distribution on* $[0, 1]$. *If we denote by $Z$ the continuous random variable defined by*

$$Z = F^{-1}(U) \tag{3.13}$$

*then $Z$ and $X$ are identically distributed.*

Therefore, if $x_0, x_1, \ldots$ is a sequence of uniformly distributed, real-valued random variables whose value is in $(0, 1)$, if $F$ is the invertible cumulative distribution function of a given continuous distribution, and if we define the sequence $x'_0, x'_1, \ldots$ as

$$\forall i \in \mathbb{N} : x'_i = F^{-1}(x_i) \tag{3.14}$$

then $x'_0, x'_1, \ldots$ is a sequence of continuous random variables that follow the distribution defined by $F$.

However, the CDF of a continuous distribution is not necessarily invertible. There exists a second method for generating continuous random variables that does not require an invertible CDF. It is called **acceptance-rejection**. We do not present this method because in this thesis, we need only to generate continuous random variates with an invertible CDF. We refer to Robert and Casella [29, Section 2.3] for a clear introduction to acceptance-rejection.

## 3.4 Output analysis and data estimation

Simulation is commonly used to estimate performance measures, avoiding the costly operations required by the explicit computation algorithms. As in Chapter 2, we are interested in two measures specific to Markov processes, namely the stationary probability vector and the expected first passage times. In this section, we show that these measures can be estimated from the simulation results. However, we must be careful when relying on simulation to compute stationary measures. The estimates can indeed be inconsistent with regard to the expected values. In this context, we give two methods that help to increase the consistency of the estimates, *i.e.* the *independent replications* and the *method of batch means*. Finally, we present the principles of the confidence intervals, which are used to give interval estimates of the measures.

### 3.4.1 Estimation of the performance measures

In this section, we present methods to estimate the stationary probabilities and the first passage times from the results of a simulation run. Thanks to these methods, we can implement the functions `setOutput` and `estimate`, which are used in the algorithm presented in Figure 3.2. The presented estimation methods are based on the intuitive definition of the measures given Chapter 2. We focus first on the stationary probability vector.

As stated in Chapter 2, the stationary probability of a given state is the probability that the process is in this specific state after a long-run time. In other words, it is the expected time spent in this state per time unit. It is also equal to the expected time spent in this state during a long observation of the process divided by the total duration of the observation. Therefore, if we observe the process behaviour thanks to a simulation, we define, for all $(k, i) \in \mathcal{S}$, the estimate of the stationary probability of state $(k, i)$ as

$$\hat{\pi}_{ki} = \frac{t_{ki}}{\sum\limits_{(h,j)\in\mathcal{S}} t_{hj}} \tag{3.15}$$

where $t_{hj}$ denotes the time spent by the simulated process in state $(h, j)$.

There is a significant difference between the effective value of vector $\hat{\pi}$ and the stationary vector we defined in Chapter 2, though. The stationary vector is a measure related to the steady state of the process. For recall, the steady state of a process is its overall behaviour after a theoretically infinite time. However, the simulation of a process behaviour also includes the transient state of the process. This implies that the simulated process requires a certain amount of time before reaching its steady state. This amount of time is called the **warm-up period**. Therefore, any estimate obtained from the simulation, in particular $\hat{\pi}$, includes simulation results related to the transient state. The influence of the warm-up period fades away over time. In particular, if we theoretically run the simulation for an infinite time, $\hat{\pi}$ would be exactly equal to stationary vector $\pi$. The key point in data estimation through simulation is therefore the epoch at which we stop the simulation, that is the difference between the overall behaviour of the simulated process and its steady state. This leads to the following three questions:

1. How to avoid the influence of the transient state?
2. How to determine the quality of the estimates, that is how close the estimates are to the real values of the considered measures?
3. How to infer the real value of a measure from the corresponding estimates?

The first question is discussed in the next subsection. The last two are dealt with in Subsection 3.4.3.

This issue is even more significant when we estimate expected first passage times. Indeed, the expected first passage time from level $k$ to level $i$ is the average first passage time from level $k$ to level $i$, obtained by taking into account all the possible behaviours. However, a simulation is merely an instance of a possible behaviour. During the simulation, we can only compute the first passage time time from level $k$ to level $i$ in the specific context of the simulated behaviour. Therefore, an expected first passage time cannot be obtained from a single simulation run. The method we present in Section 3.4.3 allows to estimate the expected first passage times from the results of multiple simulation runs.

### 3.4.2 Steady-state simulation

The main challenge in steady state simulation is the avoidance of the influence of the warm-up period in the estimates of the measures. When the warm-up period is exactly known, the problem becomes trivial. In this case, if we delay the computation of the estimates up to the end of the warm-up period, then the estimates depend only on the steady state of the process.

However, the length of the warm-up period is often a priori an unknown parameter and it is problem specific.

When the warm-up period is unknown, the influence of the transient state cannot be fully avoided. Instead, we base our estimation method on the so-called *central limit* theorem [19, Section 8.1].

**Theorem 3.4.1 (Central-limit theorem)**
*Let $X_1, X_2, \ldots, X_n$ be independent and identically distributed random variables having finite common mean $\mu$ and finite common standard deviation $\sigma$. If we denote by $\bar{X}$ the mean of this sequence, that is*

$$\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i \tag{3.16}$$

*then the mean of $\bar{X}$ is $\mu$ and its standard deviation is $\sigma/\sqrt{n}$. Furthermore, for a sufficiently large $n$, $\bar{X}$ approaches a Normal distribution with mean $\mu$ and standard deviation $\sigma/\sqrt{n}$ (see Definition 1.1.8).*

We apply this theorem to simulation-based estimation in the following example.

**Example 3.4.1** *We simulate $m$ times a discrete time QBD with the simulation algorithm given in Figure 3.2. We set the duration of each run to $n$ time units. Then, we define the set of random variables $\{X_{ij} : 0 \leq i \leq n-1, 1 \leq j \leq m\}$ such that*

- *$X_{ij} = 1$ if the process is in level 0 at the $i^{th}$ time unit of the $j^{th}$ simulation*
- *$X_{ij} = 0$ otherwise.*

*Then, we define $\bar{X}_j$, $1 \leq j \leq m$, as the estimate of $\pi_0$ given by the $j^{th}$ simulation, that is*

$$\bar{X}_j = \frac{1}{n}\sum_{i=0}^{n-1} X_{ij}. \tag{3.17}$$

*According to Theorem 3.4.1, we obtain a good estimate of $\pi_0$ by computing*

$$\bar{\bar{X}} = \frac{1}{m}\sum_{j=1}^{m} \bar{X}_j \tag{3.18}$$

Theorem 3.4.1 means that a good estimate of a performance measure is the mean of the individual estimates produced by **independent replications** of the simulation. Basically, we repeat the simulation a given number of times, modifying only the initial seeds of the random number generators. However, the use of this method leads to new issues. First, the initial state of the process clearly influences its transient state. Therefore, it also has an impact on the estimates. Another important matter is the choice of the number of replications and the duration of each of the replications. Hence, the following issues must be solved:

1. How many replications must be performed?
2. Which duration must be defined for every replication?
3. Which initial state must be chosen for all the replications?

In order to answer the first question and the second one, a compromise must be made between the quality of the estimates and the run time of the whole simulation process.

There exists an alternate method to produce such an aggregated estimate that solves the initial-state bias issue. This method is called **batch means**. Its principle is as follows. Instead of making $m$ replications with a $n$ time units duration, we make one long run with a $mn$ time units duration. We also partition the run into $m$ batches. Every batch therefore has a $n$ time units duration. For each of the batches, we compute an estimate of the performance measure. Then, we apply Theorem 3.4.1 and we compute an aggregated estimate from the estimates individually given by every batch. The batches play therefore the same role as the independent replications.

The method of batch means has several advantages. First, it clearly removes the initial-state bias, because the initial state of the process is given only to the first batch. Second, the number of batches and their individual duration do not impact on the aggregated estimate; only the total duration matters. Still, the number of batches is significant when computing *interval estimates*, as it is shown in the next section.

### 3.4.3 Confidence intervals

Thanks to the independent replications and the method of batch means, we produce a sequence of values that estimate a given performance measure. However, the transient state influences the value of these estimates. In order to reduce their impact, we apply the central limit theorem and we compute a somehow aggregated estimate that is an approximation of the real value of the measure. Yet, we have not defined a relation between the estimate and the corresponding real value of the performance measure. In particular, we present a statistical method that builds an interval that likely includes the real value of the estimated measure.

We consider a sequence $X_1, X_2, \ldots, X_n$ of independent and identically distributed random variables with common mean $\mu$ and common standard deviation $\sigma$, with $n$ being large. According to the central limit theorem, $\bar{X}$, as defined in Equation (3.16), approximatively follows a $Normal(\mu, \sigma/\sqrt{n})$ distribution. Furthermore, if we denote by $F_G$ the CDF of the $Normal(0,1)$ distribution, we have, from Definition 1.1.6 and Definition 1.1.8, that

$$P(B \leq \frac{\bar{X} - \mu}{\frac{\sigma}{\sqrt{n}}} \leq A) = F_G(A) - F_G(B) \tag{3.19}$$

In particular, when $B = -A$, this equation is equivalent to

$$P(\bar{X} - \frac{\sigma}{\sqrt{n}}A \leq \mu \leq \bar{X} + \frac{\sigma}{\sqrt{n}}A) = 1 - 2F_G(-A). \tag{3.20}$$

Next, we define $\alpha$ and $\theta$ as

$$\alpha = 2F_G(-A), \tag{3.21}$$
$$\theta = \frac{\sigma}{\sqrt{n}}F_G^{-1}(1 - \frac{\alpha}{2}). \tag{3.22}$$

Then, the last equation can be rewritten as

$$P(\mu \in [\bar{X} - \theta, \bar{X} + \theta]) = 1 - \alpha \tag{3.23}$$

By giving a value to $\alpha$, we obtain an interval that includes $\mu$ with probability $1 - \alpha$. The interval

$$[\bar{X} - \theta, \bar{X} + \theta]$$

is called **confidence interval**. Probability $1-\alpha$ is called **confidence level**. A higher confidence level means a higher guarantee that $\mu$ is in the interval, but it also increases the size of the confidence interval. In other words, a compromise must be made between the accuracy of a confidence interval and its likeliness.

It is noteworthy that $\sigma$ is generally an unknown value, while the standard deviation of the sequence $X_1, \ldots, X_n$, denoted by $s$, can be computed. It can be demonstrated that $\bar{X}$ also approximatively follows a $Normal(\mu, s/\sqrt{n})$ distribution [19, Section 8.1]. We can therefore replace $\sigma$ by $s$ in Equation (3.20).

We implement an OCTAVE function that computes a confidence interval for the expected value of a given measure. It is shown in Figure 3.3. It has two parameters, `Values` and `confidence`. `Values` is a vector containing the

```
function [inf sup] = buildConfidence(Values, confidence)
        m = avg(Values);
        s = std(Values);
        errorMargin = (1 - confidence)/2;
        theta = norminv(1 - errorMargin) * s / sqrt(n);
        inf = m - theta;
        sup = m + theta;
end
endfunction
```

Figure 3.3: An OCTAVE function that builds a confidence interval.

estimation values returned by the independent replications or the method of batch means; `confidence` is the desired level of confidence. The function returns the inferior bound and the superior bound of the corresponding confidence interval.

Using this function requires to perform a sufficiently large number of replications or batches, usually over 30. When this number is lower, a confidence interval can still be computed, though. We base its computation on the following theorem [19, Section 8.1].

**Theorem 3.4.2 (Student's test)**
*Let $X_1, X_2, \ldots, X_n$ be independent and identically distributed random variables having finite common mean $\mu$ and finite common standard deviation $\sigma$. We denote by $\bar{X}$ the mean of the sequence, and $s$ its standard deviation. Then, if we define the random variable $t$ as*

$$t = \frac{\bar{X} - \mu}{s/\sqrt{n-1}} \tag{3.24}$$

*then $t$ approximatively follows a Student(n-1) distribution.*

When the number of replications or batches is lower than 30, an algorithm similar to the one presented in Figure 3.3 can be used. However, in this case, $\theta$ is given by

$$\theta = \frac{s}{\sqrt{n-1}} F^{-1}_{S(n-1)}(\frac{\alpha}{2}) \tag{3.25}$$

where $F_{S(x)}$ denotes the Student CDF with $x$ degrees of freedom.

It is noteworthy that if the number of replications or batches is low, and if the influence of the transient state is too strong, then the confidence interval is large and therefore inaccurate. When it happens, we recommend to start again the simulation after increasing the simulation time. This way, the influence of the transient state is reduced.

## 3.5 Conclusion

Through this chapter, we introduced a simulation approach and we applied it to discrete time QBDs. After we developed the simulation algorithm, we focused on the random number generation and the analysis of the simulation results. In particular, we presented an estimation method for computing a confidence interval that includes the expected value of a given measure.

We end the chapter by giving the adaptations needed by the algorithm shown in Figure 3.2 to simulate a continuous time QBD. The new algorithm we develop is based on the uniformization technique and its interpretation. Basically, the QBD is uniformized with a rate $c$ satisfying Equation (1.37). The resulting transition matrix, denoted by $K$, is used to determine at each iteration the next state of the process. $K$ therefore plays the same role as the transition matrix $T$ in the discrete case. However, the clock is not increased by exactly one unit at each iteration. Instead, we generate a random variable following an exponential distribution with rate $c$ and we add its value to the current clock time. Hence, this simulation algorithm agrees with the definition of the uniformization we gave in Section 1.3.3. It is worth mentioning that the value of $c$ has a significant influence on the run time of the algorithm. Indeed, according to the definition of the exponential distribution, if we use a uniformization rate as low as possible, the clock time increases more rapidly. Therefore, the number of iterations needed to reach the end of the simulation is reduced.

# Part III

# Tool implementation

# Chapter 4

# QBDSense Tool

In the previous chapters, we presented two approaches that provide methods to evaluate the performance of a QBD process. In the present chapter, we make intensive use of these methods to build up our own software, that we call QBDSense.

There already exist tools that implement algorithms issued from the matrix analytic methods. For example, MAGIC [30] and MGMTool [12] provide solutions to compute the steady state probabilities of a homogeneous QBD process. Two other tools exist: MAMSolver [28] and SMCSolver [2]. MAMSolver extends the analysis to M/G/1 and GI/M/1 cases. Its implementation is based on the ETAQA methodology [5]. SMCSolver provides some functions that compute the essential matrices $G$, $R$ and $U$, as well as the steady state probability vector of a homogeneous QBD process.

However, all of these tools have several limitations we want to overcome. First, they do not permit the analysis of inhomogeneous QBDs. Second, they are not concerned with the difficulties inherent to the definition of the matrices composing the transition matrix or the infinitesimal generator of the process. Existing scientific literature confirms that some QBDs exhibit a transition matrix or a generator with infinite size or with huge sized inner blocks (see *e.g.* the case studies carried out by Hautphenne *et al.* [11], and Montoro and Perez [23]). Furthermore, their analyses are limited to the original process itself and they do not evaluate the sensitivity of the performance measures with regard to a given perturbation. Finally, none of them provides a simulation framework that allows to approximate the value of the performance measures instead of exactly computing it using matrix analytic methods.

In this chapter, we present a tool we developed that gets rid of those restrictions. We call that tool QBDSense. It offers three different input interfaces to specify a Quasi-Birth-and-Death process, which can be either homogeneous or inhomogeneous. We also allow the specification of the QBD to be parameterised in order to perform sensitivity analyses on performance measures. The results can be computed either by using implementations of the matrix analytic methods, or by applying simulation algorithms based on the discrete-event simulation approaches.

The chapter is presented as follows. Section 4.1 presents the general structure and the functionalities of the tool. The guidelines to perform an analysis are given in Section 4.2. This section is especially focused on how to specify a Quasi-Birth-and-Death process. Section 4.3 describes how the implementations of the performance analysis algorithms are provided by the tool. Next, Section 4.5 presents more particularly the architecture of the tool. Finally, Section 4.6 discusses the limitations of the tool as well as some perspectives to overcome these ones.

This chapter is related to two of our own publications, namely Cordy and Remiche [6, 7]. These publications are given in Appendices A and B, respectively.

## 4.1 General architecture and functionalities

QBDSense can be seen as the collection of individual components that communicate to perform a complete analysis. There are mainly three types of components:

1. The **user interfaces** are composed of three **input interfaces** and several **output interfaces**.

   - The input interfaces consist in three different methods to specify a QBD. Each one has advantages and drawbacks. The user can therefore choose the most appropriate method to specify the process he wants to evaluate.
   - The output interfaces display the result of the analyses either as a text or as charts.

2. The **analysis frameworks** compute the performance measures. We can distinguish two frameworks, namely the **matrix analytic methods** framework and the **simulation** framework.

   - The matrix analytic methods framework is composed of implementations of the algorithms presented in Chapter 2.
   - The simulation framework provides general procedures to simulate a QBD process. Accordingly, it estimates some performance

measures and it offers statistical tests to evaluate the quality of the estimations. It is based on the simulation approaches discussed in Chapter 3.

3. The **linkers** are intermediary components that link together the user interfaces and the analysis frameworks.

Although they are part of QBDSense, some of the input interfaces and the analysis frameworks are individually provided as a stand-alone resource. Therefore, any of them can be used independently from the others. This property facilitates the integration with third-party software components.

We describe QBDSense from the point of view of the final user; that is we explain all the possibilities offered to specify a QBD process and to evaluate its performance. To achieve this, the user must use the tool as follows:

1. Once the tool is opened, one of the input interfaces must be selected. Some of these interfaces require a specific file that the user must provide. We explain more precisely the content of such a file later.
2. Then, the type of analysis to perform must be chosen, that is either an ordinary performance analysis or a sensitivity analysis.
3. A computation method must be selected, namely the matrix analytic methods or the simulation approach.
4. Once the computation of the performance measure is complete, the performance measures to display must be chosen.

In the next section, we propose to explain the input interfaces, that is which parameters must be provided in order to perform an analysis. We also present how the user can define a perturbation on the process to evaluate. Next, Section 4.3 is devoted to the presentation of the analysis frameworks. Section 4.4 describes how the analysis results are presented by the output interfaces. Finally, Section 4.5 gives a technical description of the architecture, in particular the intermediary components we call linkers.

## 4.2 Description of the input interfaces

Our tool provides three different methods to specify a QBD process: this process can be specified (i) by giving its infinitesimal generator, (ii) as a type of queue using Kendall's notation in concise form, or (iii) by declaring its possible transitions. In this section, we explain first each of these specification methods. As an example, we illustrate them to define a QBD that models the following M/PH/1 queueing system.

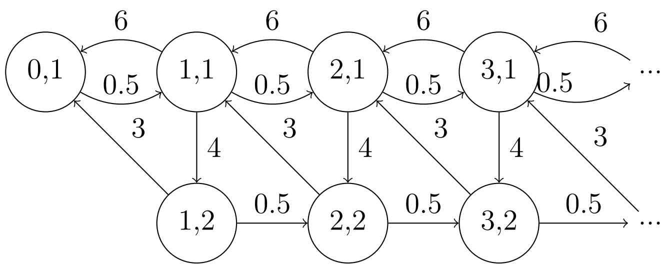


Figure 4.1: The possible transitions in the M/PH/1 queueing system of Example 4.2.1.

**Example 4.2.1** *We consider a M/PH/1 queueing system with an infinite capacity. The time between two consecutive arrivals follows an exponential distribution with parameter $\lambda$, to which we give a value of 0.5. The service time follows a $PH(\alpha, T)$ distribution where:*

$$\alpha = \begin{bmatrix} 1 & 0 \end{bmatrix},$$
$$T = \begin{bmatrix} -10 & 4 \\ 0 & -3 \end{bmatrix}.$$

*We model the queueing system with a QBD process with state space defined as*

$$\mathcal{S} = \{(0,1)\} \cup \{(k,i) : k \in \mathbb{N}_0, i \in \{1,2\}\}. \tag{4.1}$$

*The process is in level 0*[1] *when the system is empty. Otherwise, it is in level $k$ and in phase $i$ when the current number of jobs in the system is $k$ and when the phase of the current service time is $i$. Therefore, the possible transitions of the process are, for all $k > 0$ and $1 \leq i \leq 2$:*

- *from $(0,1)$ to $(1,1)$ with a rate of 0.5,*
- *from $(k,i)$ to $(k+1,i)$ with a rate of 0.5,*
- *from $(k+1,1)$ to $(k+1,2)$ with a rate of 4,*
- *from $(k+1,1)$ to $(l,1)$ with a rate of 6, and*
- *from $(k+1,2)$ to $(l,1)$ with a rate of 3.*

*A graph showing these transitions is given in Figure 4.1.*

The three methods are now given. For each of these methods, we explain how to specify a Quasi-Birth-and-Death process.

[1]Level 0 has only one phase, which is phase 1.

### 4.2.1 Method 1: Explicit definition of the inner blocks

The first method consists in explicitly giving the inner blocks that compose the infinitesimal generator of the process. For this purpose, the user must provide an OCTAVE function, which must be written in a **.m** file. This function must respect a particular syntax. Indeed, the function signature must include one parameter, that is the level we want to compute the blocks of. The return values of the function are the three inner blocks of the generator $Q$ corresponding to the specified level. The computed inner blocks are returned in the following order: $A_{-1}^{(i)}$, $A_0^{(i)}$ and $A_1^{(i)}$, such as they are defined in Equation (1.21). If one of these matrices is not defined for the considered level, it must be replaced by an empty matrix. For example, the result of the function for the level 0 would be: [], $B_0$, $B_1$.

Figure 4.2 shows an OCTAVE function that specifies the system presented in Example 4.2.1. First, we use variables to record the parameters of the system. Next, we build the matrices to return depending on the level given as parameter. Of course, the matrices returned by the function are exactly similar to the inner blocks composing the infinitesimal generator of an infinite M/PH/1 queueing system, as we have showed in Section 1.3.2.

Once the function is written in the **.m** file, the user must specify the absolute path to that file in the graphical user interface. Thanks to this specification method, we can define any QBD process. However, it requires to program in OCTAVE and to know the exact form of the infinitesimal generator, which can be particularly laborious. The other input methods intend to overcome these drawbacks.

### 4.2.2 Method 2: Definition of a queueing system

The second method allows to define the QBD process as a queueing system, using the concise form of Kendall's notation (see Subsection 1.3.4). After choosing the queue type, each parameter of the selected system, such as an exponential rate or the parameters of a phase-type distribution, must be entered. The current version of the tool only supports the $M/M/1$, $M/PH/1$, $PH/M/1$ and $PH/PH/1$ queues with a *first in first out* policy and no priority. The queue capacity can be selected as finite or infinite. In the former case, the exact capacity must be given. Figure 4.3 shows the graphical interface that permits the user to define the queueing system presented in Example 4.2.1. The user must introduce the value of the parameters $\lambda$, $\alpha$ and $T$. Also, the queue capacity must be specified as infinite. As we will explain in Section 4.4, a number of levels must be chosen in order to limit the display of the results. It is noteworthy that the syntax to specify a vector or

```
[A_1 A0 A1] = getInnerBlocks(level)
        lambda = 0.5;
        alpha = [1 0];
        T = [-10 4; 0 -3];
        t = -sum(T')';
        if(level == 0)
                A_1 = [];
                A0 = -lambda;
                A1 = lambda * alpha;
        elseif (level == 1);
                A_1 = t;
                A0 = T - lambda * eye(size(T));
                A1 = lambda * eye(size(T));
        else
                A_1 = t * alpha;
                A0 = T - lambda * eye(size(T));
                A1 = lambda * eye(size(T));
        end
endfunction
```

Figure 4.2: An Octave function that computes the inner block composing the generator of a QBD modeling an M/Ph/1 queue.

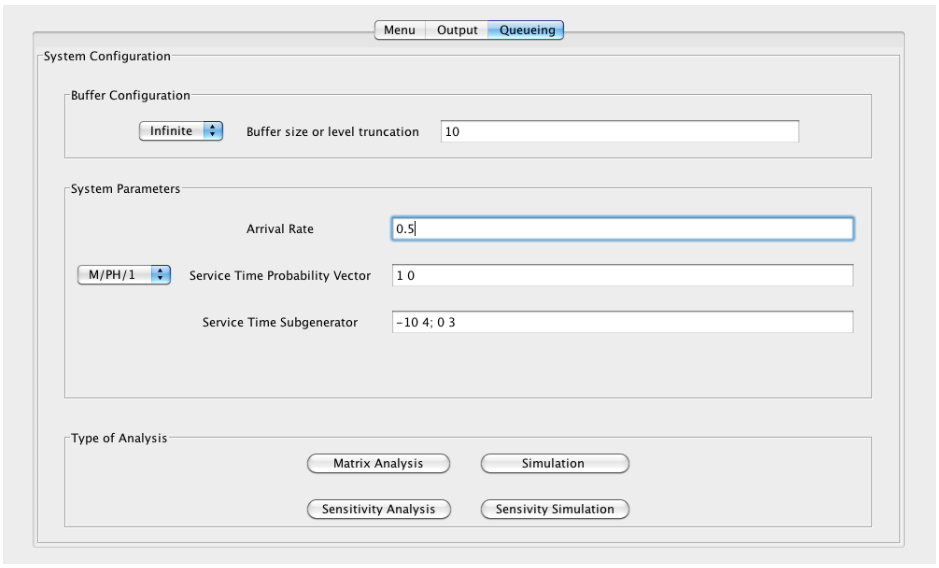
a matrix is similar to Octave. Once the user has specified all the required parameters, the tool automatically computes an Octave function similar to the one the user would have implemented if he used the first specification method.

### 4.2.3 Method 3: Specification of the transitions

The third and last input method consists in specifying the possible transitions of the QBD process in a syntax-consistent and textual language we developed. The text defining the QBD must be written in a **.qbd** file. The particular feature of our method is that all the possible transitions have not to be given. Instead, the user can specify more concisely a repetitive structure appearing in the generator.

QBDParser is a Java library we developed in order to produce, thanks to a text file describing the possible transitions of a QBD, an Octave function that computes the inner block of a given level of this QBD. Basically, the library is composed of:

- a parser that is able to read and “understand” a **.qbd** file containing a textual description of the transitions,
- some data structures used to record the transitions analysed by the parser, and



Infinite Buffer size or level truncation 10

Arrival Rate 0.5

M/PH/1 Service Time Probability Vector 1 0

Service Time Subgenerator -10 4; 0 3

Figure 4.3: A graphical user interface used to specify a QBD as a queueing system.

- a code producer that writes an OCTAVE FUNCTION thanks to the content of the data structures. This function can then be used to obtain the inner blocks composing the generator of a QBD.

Once the **.qbd** file is written, the user must specify the absolute path to that file. Then, QBDPARSER parses the file and automatically produces the corresponding OCTAVE function, which is used by the performance analysis frameworks afterwards.

We built the parser thanks to **JavaCC**[2], an open source tool originally owned by Sun Microsystems. Basically, this tool helps to generate a lexical analyser and a descending syntax analyser from a context-free grammar defined in *Backus-Naur Form*. We do not cover the theory about lexical and syntactical analysis and we refer to Aho *et al.* [1] for a clear introduction.

Next, we focus on the language we use to specify the transitions of the QBD. First, we describe its syntax. Then, the associated semantic is given.

### Syntax of the specification language

In this section, we present the syntax of the language. Every element of the syntax in given in *Backus-Naur Form*. A formal definition of the semantic

[2]https://javacc.dev.java.net/

is given afterwards.

```
qbd ::= [declaration]* definition+
```

The specification of a QBD process is composed of a series of constant declarations and a series of transition definitions. No constant can be declared but there must be at least one transition definition.

```
declaration ::= 'CONST' id '=' value ';'
id ::= ['a'-'z'] (['a'-'z', 'A'-'Z', '0'-'9', '_'])*
value ::= ('0' | (['1'-'9'] (['0'-'9'])*)) (['.'](['0'-'9'])+)
```

The declaration of a constant begins with the keyword `CONST`. It is followed by an *id*, *i.e.* a string beginning with a lower case letter, the sign `=` and a real number. A declaration ends with a semi-colon (`;`). Informally, a constant declaration associates a value with a string. Whenever this string is used in a transition definition, it is considered as the value associated with it.

```
declaration ::= 'TO' '(' type ',' expression ')'
                'FOR' conditions 'RATE' expression ';'
type ::= 'UP' | 'SAME' | 'DOWN'
```

A transition definition begins with the keyword `TO`. It is followed by the type of transition and the destination phase. More precisely,

- the possible types of transition are `UP`, `DOWN` and `SAME`, which describe if the process moves to the upper level, to the downer level or to the same level, respectively, and
- the destination phase is given by an `expression`. The syntax of an expression is given a bit further.

Then, the keyword `FOR` precedes `conditions` specifying the states from which the transition is available. Finally, the keyword `RATE` is followed by an expression specifying at which rate the transition occurs.

```
conditions ::= condition ['AND' condition]*
condition ::= macrocondition | microcondition
macrocondition ::= 'L' ('<' | '>' | '=' | '<=' | '>=') value
microcondition ::= 'P' ('<' | '>' | '=' | '<=' | '>=') value
```

The conditions specifying the state from which a transition can be fired are separated by the keyword `AND`. A `condition` is either a `macrocondition` or a `microcondition`. A `macrocondition` is an arithmetic constraint defined on the current level, represented by the keyword `L`. A `microcondition` is also an arithmetic constraint, but it is defined on the current phase, represented by the keyword P.

```
expression ::= expression '+' expression
         | expression '-' expression
         | expression '*' expression
         | expression '/' expression
         | '-' expression
         | 'min' '(' expression ',' expression ')'
         | 'max' '(' expression ',' expression ')'
         | '(' expression ')'
         | 'L'
         | 'P'
         | id
         | value
```

An `expression` can be either a composite expression or a simple expression. A composite expression is made from the following arithmetic operators: addition (`+`), subtraction (`-`), multiplication (`*`), division (`/`). It can also be the minimum (`min`) and maximum (`max`) functions. Brackets can also be used to define an order on the expressions. A simple expression can be a `value`, the current level `L`, the current phase `P` or an `id`, namely a reference to a previously declared constant. It is worth mentioning that thanks to the use of the variables `L` and `P`, we can concisely define some inhomogeneous QBDs.

In Figure 4.4, we illustrate the use of this specification method applied to Example 4.2.1. The parameters of the system are declared as constants (see lines 1-6). Line 7 describes a transition to the phase 1 of the upper level. As the following constraints express, this transition is available only when the process is in the phase 1 of the level 0. The rate of the transition is `lambda`, which has a value of 0.5 according to the constant declaration at line 1. This transition corresponds to the only transition that is available from level 0 (see Figure 4.1). Line 8 defines the transition of rate 4 available from any state $(k+1,1), k \geq 0$, to state $(k+1,2)$. Line 9 specifies the transition of rate 6 available from any state $(k+1,1), k \geq 0$, to state $(k,1)$. Line 10 gives the transition of rate 3 available from every state $(k+1,2)$ to state $(k,1)$. Finally, line 11 defines the transition of rate 0.5 that goes from any state $(k,i), k \geq 0, i \in \{1,2\}$ to state $(k+1,i)$.

Next, we define the semantic associated with the language.

## Semantic of the language

The semantic of a constant declaration is merely a mapping between a string and an arithmetic value. The semantic of a transition definition is a data structure we define hereafter.

```
CONST lambda = 0.5;
CONST mu1 = 10;
CONST p = 0.6;
CONST q = 0.4;
CONST mu2 = 3;

TRANSITION (UP,1)         FOR L = 0 AND P = 1       RATE lambda;
TRANSITION(SAME, P+1)     FOR L > 1 AND P = 1       RATE mu1*q;
TRANSITION(DOWN, 1)       FOR L > 1 AND P = 1       RATE mu1*p;
TRANSITION(DOWN,1)        FOR L > 1 AND P = 2       RATE mu2;
TRANSITION(UP,P)          FOR L>0 AND P>0 AND P<3 RATE lambda;
```

Figure 4.4: Specification of a QBD with a textual language.

For recall, our objective is to create, from a text written in the syntax we have defined, an OCTAVE function that builds the inner blocks of any level of the process. To achieve this, data structures we call *Similar Macrostates Group* (*SMG*) must be built during the parsing. The SMGs are formally defined as follows.

**Definition 4.2.1 (Similar Macrostates Group)**
*A Similar Macrostates Group is a triplet (L, d, M) where:*

- *$L$ is a no empty set of levels of the QBD,*
- *$d \in \{-1, 0, 1\}$ gives a type of transition*
- *$M$ is a no empty set of triplet (P, t, r) where:*
  - *$P$ is a no empty set of phases that are defined in every level $k \in L$,*
  - *$t$ is an arithmetic expression that defines the targeted phase, and*
  - *$r$ is an arithmetic expression that defines the rate of the transition.*

A SMG defines a set of states $\{(k, i)\}$ such that the process can move, from every state in this set, to state $t$ with rate $r$ where $t$ and $r$ are determined by computing the value of given arithmetic expressions. In order to take into account the repetitive structure of a QBD, these expressions can contain the symbolic values $k$ and $i$. They can also include references to the declared constants. More formally, if we denote by $s \xrightarrow{\tau} s'$ that a transition of rate $\tau$

is available from state $s$ to state $s'$, we have, for a given SMG $(L, d, M)$ that:

$$\forall k \in L, \forall (P, t, r) \in M : \forall i \in P : (k, i) \xrightarrow{r} (k + d, t) \tag{4.2}$$

The semantic of every transition defined with our grammar is a SMG. We illustrate this statement in the following example.

**Example 4.2.2** *The semantic of the transitions defined at lines 9 and 10 in Figure 4.4 is respectively two SMGs $(L, d, M)$ and $(L', d', M')$ such that*

$$L = \{k : 1 < k < +\infty\} \qquad L' = \{k : 1 < k < +\infty\} \tag{4.3a}$$

$$d = -1 \qquad d' = -1 \tag{4.3b}$$

$$M = \{(\{1\}, 1, 10 * 0.4)\} \qquad M' = \{(\{2\}, 1, 3)\} \tag{4.3c}$$

It is noteworthy that each type of transition (*i.e.* to the upper level, to the downer level and to the same level) defines some elements of a different type of inner block. Therefore, any SMG $(L, d, m)$ corresponds with a given set of inner blocks. More precisely,

- if $d = -1$, the corresponding set of blocks is $\{A_{-1}^{(k)} : k \in L\}$;
- if $d = 0$, it is $\{A_0^{(k)} : k \in L\}$; and
- if $d = 1$, it is $\{A_1^{(k)} : k \in L\}$.

This implies that two SMGs $(L, d, M)$ and $(L', d', M')$, with $d \neq d'$, do not define elements of the same inner blocks.

Accordingly, we define three lists of SMGs, each of them corresponding to a type of transition. Initially, these lists are empty. At the end, we want each list to contain SMGs such that the intersection of the set of levels of two distinct SMGs is empty. In other words, we want the following property:

$$\forall S = (L, d, M), S' = (L', d', M') \in \mathit{list} : S \neq S' : L \cap L' = \emptyset \tag{4.4}$$

Before we present how to construct the three lists, we have to define the *fusion* of two *SMGs*. Informally, the fusion gathers the common levels into a new *SMG* with all the transitions and removes those levels from the original SMGs. If a resulting SMG has an empty set of levels, it is discarded. More formally, we define it as follows.

**Definition 4.2.2 (Fusion of two Similar Macrostates Groups)** *If we denote by $SSMG$ the set of all possible $SMGs$, the **fusion** of two $SMGs$ is a function*

$$fuse : SSMG \times SSMG \longrightarrow \mathcal{P}(SSMG)$$

*where $\mathcal{P}$ denotes the powerset, such that:*

$$\forall S = (L, d, M), S' = (L', d', M') \in SSMG :$$

- *if $d \neq d' \vee L \cap L' = \emptyset : fuse(S, S') = \{S, S'\}$*
- *else if $L = L' : fuse(S, S') = \{(L, d, M \cup M')\}$*
- *else if $L' \subset L : fuse(S, S') = \{(L \setminus L', d, M), (L', d, M \cup M')\}$*
- *else if $L \subset L' : fuse(S, S') = \{(L, d, M \cup M), (L \setminus L', d, M)\}$*
- *else : fuse(S, S') = $\{(L\setminus L', d, M)(L\cap L', d, M\cup M'), (L'\setminus L, d, M')\}$*

We illustrate the use of the fusion function in the following example.

**Example 4.2.3** *If we apply this function at the two SMGs defined in Example 4.2.2, the result would be a set that contains a single $SMG$ $(L'', d'', M'')$ where*

$$\begin{aligned} L'' &= \{k : 1 < k < +\infty\} \\ d'' &= -1 \\ M'' &= \{(\{1\}, 1, 10 * 0.4), (\{2\}, 1, 3)\} \end{aligned}$$

With this operator, we can build the three lists provided that the transitions defined in the text file are consistent. This means that, for two given states $s$ and $s'$, two transitions from $s$ to $s'$ with a different rate cannot exist. Indeed, if the hypothesis holds, the fusion operator clearly preserves the consistency of the transitions defined in the original $SMGs$.

The construction of the lists is as follows. Each time the parser parses the definition of a transition, the corresponding $SMG$, $S = (L, d, M)$, is built. Then, we get the SMGs list corresponding to the transition type, which we denote by $l$. If $l$ is empty, $S$ is immediately added to it. Otherwise, we execute the following steps.

1. We define *temp* as an empty list.
2. We define $S'$ as the first $SMG$ of $l$ and we remove $S'$ from $l$.

3. We apply the fusion operator to $S'$ and $S$.
4. The resulting SMG that corresponds with levels $k \in L \cap L'$ and the one that corresponds with levels $k \in L' \setminus L$, if they exist, are added to *temp*.
5. - If $L \setminus L' = \emptyset$, we add the elements of *temp* to $l$ and we stop.
   - Otherwise, $S$ becomes the SMG resulting from the fusion that corresponds with levels $k \in L \setminus L'$.
6. - If $l$ is empty, $S$ is added to *temp*, $l$ becomes *temp*, and we stop.
   - Otherwise, we define $S'$ as the current first element of $l$, we remove $S'$ from $l$, and we go back to step 3.

At the end, each level of the process is referenced at most one time in each list. From this point on, an OCTAVE function that computes the inner blocks can easily be produced. For recall, each list corresponds with one of the three matrices that must be returned by the OCTAVE function. We illustrate the code generation in the following example.

**Example 4.2.4** *We consider the transition specification shown in Figure 4.4. We want to compute the inner blocks related to the transitions to the lower level, namely the matrices $A_{-1}^{(k)}$. Lines 9 and 10 specify such a transition. We denote by $l$ the list of SMGs we want to build, which is initially empty.*

*Line 9 is parsed first and the corresponding SMG is built, as we did in Example 4.2.2. We denote this SMG by $S = (L, -1, M)$. Because $l$ is empty, $S$ is immediately added to it.*

*Next, Line 10 is parsed and an SMG $S' = (L', -1, M')$ is built accordingly. This time, $l$ is not empty, and we remove its first element (namely $S$). We define temp as an empty list. We apply the $fuse$ operator to $S$ and $S'$. As in Example 4.2.3, it results a single SMG*

$$S'' = (L'', -1, M'') \tag{4.5}$$

*where*

$$\begin{aligned} L'' &= \{k : 1 < k < +\infty\}, \\ d'' &= -1, \\ M'' &= \{(\{1\}, 1, 4), (\{2\}, 1, 3)\}. \end{aligned}$$

*It is noteworthy that*

$$L'' = L \cup L'. \tag{4.6}$$

*$S''$ is added to temp. Finally, according to step 5, $l$ becomes temp because*

$$L \setminus L' = \emptyset. \tag{4.7}$$

*We have finished to built the list.*

*To produce the* Octave *function, we successively get the elements of $l$. In our example, there is only one element, namely*

$$S'' = (L'', -1, M''). \tag{4.8}$$

*For all $k \in L$", the set*

$$M'' = \{(P'', t'', r'')\} \tag{4.9}$$

*defines some strictly positive elements of matrix $A_{-1}^{(k)}$. More precisely,*

- *each element of $P$ refers to exactly one row of the matrix;*
- *the target expression $t$ refers to exactly one column of the matrix;*
- *the rate expression $r$ gives the value of the corresponding elements of the matrix.*

*According to our example, $M$" defines the following elements of $A_{-1}^{(k)}$:*

$$(A_{-1}^{(k)})_{1,1} = 4$$
$$(A_{-1}^{(k)})_{2,1} = 3$$

The two other types of inner blocks can similarly be computed. Matrices $A_0^{(k)}$ are particular though. Because such a matrix is the central inner block, each ith diagonal element must be equal to

$$-(\sum_{j=1}^{n_{k-1}} (A_{-1}^{(k)})_{ij} + \sum_{j=1}^{n_k} (A_0^{(k)})_{ij} + \sum_{j=1}^{n_{k+1}} (A_1^{(k)})_{ij}) \tag{4.10}$$

where $n_k$ denotes the number of phases in level $k$.

Finally, it is worth mentioning that, if necessary, a block must be resized by adding some lines and columns filled with zeros. For example, if we consider an inhomogeneous and infinite QBD and if we denote by $row(M)$ and $col(M)$, respectively, the number of rows and columns of matrix $M$, the following properties must hold:

$$\forall k, row(A_0^{(k)}) = row(A_1^{(k)}) \tag{4.11}$$
$$\forall k > 0, row(A_0^{(k)}) = row(A_{-1}^{(k)}) \tag{4.12}$$
$$\forall k, row(A_1^{(k)}) = col(A_{-1}^{(k+1)}) \tag{4.13}$$

By way of illustration, we must add a second column filled with zeros to matrix $A_{-1}^{(k)}$ generated in Example 4.2.4.

### 4.2.4 Parameterisation of a sensitivity analysis

After using one of the methods to specify a QBD, the user must select the kind of analysis he wants to perform. More precisely, the possible types of analysis are:

1. a performance analysis, which consists in the computation of the stationary vector and the expected first passage times, and
2. a sensitivity analysis, *i.e.* the measurement of the evolution of the performance measures with regard to small variations applied to the process parameters.

Performing a sensitivity analysis requires the definition of the perturbation to apply to the QBD. More precisely, we define the perturbed generator to analyse as

$$\tilde{Q}(\epsilon) = Q + P(\epsilon) \tag{4.14}$$

where we successively give different values to $\epsilon$. Hence, for each value given to $\epsilon$, we compute the performance measures related to $\tilde{Q}(\epsilon)$ in order to observe their evolution according to the perturbation. We present how this evolution is displayed in Section 4.4. We now describe how to define this perturbation with each of the three input interfaces.

The first method, *i.e.* the explicit implementation of an OCTAVE function, easily permits to define a perturbation encountered by the QBD. For that purpose, the function declaration must include a second parameter, that is $\epsilon$. This parameter – we call it variable of perturbation – can influence the value of any inner block.

**Example 4.2.5** *Figure 4.5 shows how to modify the function presented in Figure 4.2 in order to define a perturbation on the arrival rate of the system presented in Example 4.2.1. The only differences are the addition of a second parameter in the declaration of the function and the modification of the value assigned to* `lambda` *using this parameter.*

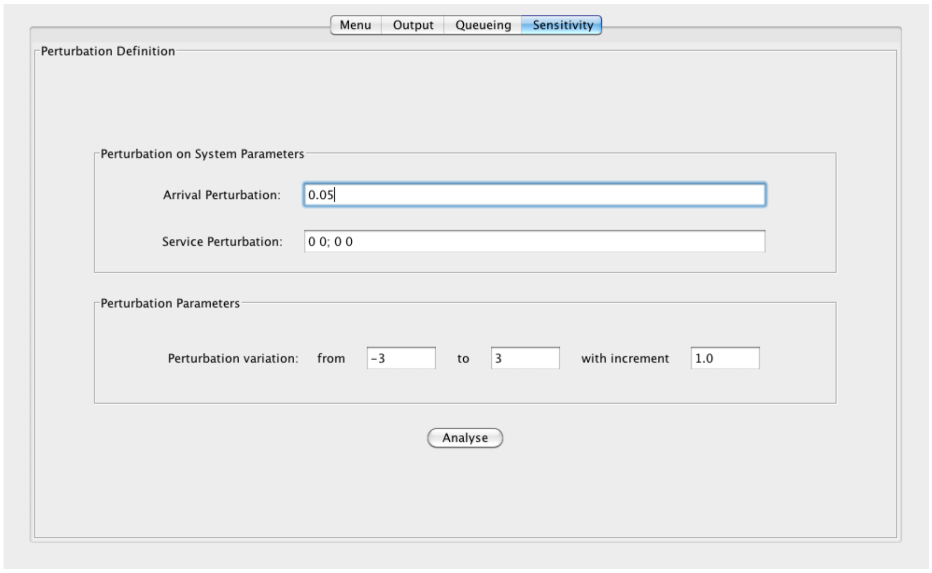
The second method allows to apply a perturbation to the defined queueing system. For each initial parameter $p$ of the original queueing system, the user must specify an additional parameter $p'$. Then, the value of $p$ becomes $p + \epsilon p'$. As before, $\epsilon$ is the variable of perturbation.

**Example 4.2.6** *Figure 4.6 illustrates the definition of a perturbation on the arrival rate of the system defined in Example 4.2.1.*

Finally, a perturbation on the QBD can also be defined with the last method by extending the grammar of the language. More precisely, we add

```
[A_1 A0 A1] = getInnerBlocks(level, e)
        lambda = 0.5 + e * 0.05;
        alpha = [1 0];
        T = [-10 4; 0 -3];
        t = -sum(T')';
        ...
endfunction
```

Figure 4.5: An OCTAVE function that computes the inner blocks composing the generator of a QBD modelling a **perturbed** M/Ph/1 queue.



Perturbation Definition

Perturbation on System Parameters

Arrival Perturbation: 0.05

Service Perturbation: 0 0; 0 0

Perturbation Parameters

Perturbation variation: from -3 to 3 with increment 1.0

Figure 4.6: A graphical user interface used to define a perturbation on a queueing system.

the possibility for a simple `expression` to be the variable `E`, which represents the variable of perturbation. We can then use this variable to modify a transition rate with respect to the value that will be given to `E`.

**Example 4.2.7** *We want to apply a perturbation of* $\epsilon * 0.05$ *on the arrival rate of the system presented in Example 4.2.1. Then, we must modify only the first line of the text file whose we gave the content in Figure 4.4. After the modification, this line would be:*

```
CONST lambda = 0.5 + E * 0.05;
```

| Function | Output | QBD types |
|---|---|---|
| `QBD_FPT` | $\mathbf{t}_{i\to j}$ | Hom. & Inf. |
| `QBD_FPT_Finite` | $\mathbf{t}_{i\to j}$ | Hom. & Fin. |
| `QBD_FPT_Inh` | $\mathbf{t}_{i\to j}$ | Inh. |
| `QBD_LR` | $G, R, U$ | Hom. & Inf. |
| `QBD_pi` | $\pi$ | Hom. & Inf. |
| `QBD_pi_finite` | $\pi$ | Hom. & Fin. |
| `QBD_pi_Inh` | $\pi$ | Inh. |

|  |  |
|---|---|
| Hom. : Homogeneous | Inh. : Inhomogeneous |
| Fin. : Finite | Inf. : Infinite |

Table 4.1: The main Octave functions included in the framework QBDSolver.

## 4.3 The performance analysis frameworks

By using the input interfaces, the user can specify a QBD. Next, the performance of the process is analysed thanks to one of two frameworks we developed: QBDSolver and QBDSimulator. While the former makes use of the matrix analytic methods, the latter simulates the QBD in order to estimate the performance measures. We successively describe each of the frameworks.

### 4.3.1 QBDSolver

QBDSolver is a framework that allows the analysis of a QBD process using the matrix analytic methods. It is a collection of Octave functions. Some of these functions are part of the *SMCSolver* developed by Bini *et al.* [2]. The others implement algorithms presented in Chapter 2. Thanks to this collection, we can evaluate every type of QBD. All these functions can be either called from the Octave command line or from our tool. We describe the most important functions. Table 4.1, associates the functions with the value they compute and with the types of QBD to which they are applied.

The first three functions compute the expected first passage times from every state of a given level (denoted by $i$) to another level (denoted by $j$). Each of the three functions can only be applied to a specific type of QBD. More precisely,

- the first function can be applied to a homogeneous and finite QBD,
- the second one can be used when the QBD is homogeneous and finite,
- the third one is intended for an inhomogeneous QBD.

They all require the same two parameters: $i$ and $j$, respectively. The functions dynamically gets the inner blocks of a given level by calling a function called `getInnerBlocks`, which returns the three inner blocks related to a level given as parameter. Basically, this function must be either provided by the user of the framework or produced by one of the input interfaces we described in Section 4.2.

The fourth function computes the rate matrix of a homogeneous and infinite QBD, as well as the two related matrices $G$ and $U$ (see Chapter 2.1.3). It requires the following parameters: $A_{-1}, A_0, A_1$, as defined in Equation (1.24). This function comes from the SMCSolver of Bini *et al.* [2].

Finally, the last three functions computes $\pi$, the stationary vector of a specific type of QBD. More precisely, `QBD_pi` computes a subvector of $\pi$ for a homogeneous and infinite QBD. Its parameters are respectively: $B_{-1}, B_0, B_1, A_{-1}, A_0, R$, and the maximum level of which the stationary probability is computed. This function also comes from the SMCSolver. `QBD_pi_finite` computes the whole stationary vector of a finite and homogeneous QBD. It requires the following parameters: $B_{-1}$, $B_0$, $B_1$, $A_{-1}$, $A_0$, $A_1$, $C_{-1}$, $C_0$, $C_1$ and the maximum level reachable by the process. Finally, `QBD_pi_Inh` is intended for inhomogeneous QBDs. It requires only one parameter, namely the maximum level of which the stationary probability must be computed. It dynamically computes the inner blocks it needs by calling the function `getInnerBlocks`.

These functions are provided as a stand-alone resource, so that we can call them directly with Octave, instead of observing their results via the QBDSense tool. Furthermore, it allows any third-party developer to implement more efficient algorithms or to extend the framework by providing implementation of different algorithms.

### 4.3.2 QBDSimulator

When evaluating a QBD composed of huge inner blocks, the run time of the matrix analytic methods algorithms can be significant. Indeed, they are based on matrix multiplication and matrix inverse, which are costly operations. Therefore, we provide another framework that permits to estimate the value of some interesting measures instead of computing it exactly. This estimation is done thanks to the discrete-event simulation algorithm presented in Chapter 3. This algorithm has been implemented in Octave functions that are part of the framework QBDSimulator. We present the simulation functions in Table 4.2.

The function `QBD_sim_confidence` computes a confidence interval for a

| Function | Output | QBD types |
|---|---|---|
| `QBD_sim_confidence` | Confidence interval | N/A |
| `QBD_sim_hom` | $\hat{\pi}, \mathbf{t}_{i \to j}$ | Hom. & Inf. |
| `QBD_sim_hom_fin` | $\hat{\pi}, \mathbf{t}_{i \to j}$ | Hom. & Fin. |
| `QBD_sim_inh` | $\hat{\pi}, \mathbf{t}_{i \to j}$ | Inh. |

| | |
|---|---|
| Hom. : Homogeneous | Inh. : Inhomogeneous |
| Fin. : Finite | Inf. : Infinite |

Table 4.2: The main OCTAVE functions included in the framework QBD-SIMULATOR.

given measure. Its parameters are respectively: `Samples`, a vector containing the estimates value, and `level`, the confidence level of the interval to compute. The function returns the inferior bound and the superior bound of the interval, respectively.

The other three functions simulate a QBD. They require the same parameters, which are:

- the level and the phase in which the process starts,
- the length of the warm-up period (can be set to zero if it is unknown),
- the simulation end time,
- the number of levels of the process,
- two numbers defining the seeds used by the two random number generators we use[3], and
- the number of batches in which the simulation is divided (see Section 3.4.2).

When the number of batches is greater than one, then the method of batch means is used to estimate the performance measure.

It is worth mentioning that the three functions dynamically compute the inner blocks of the QBD by calling the function `getInnerBlocks`. This call has a significant influence on the run time of the algorithm. Therefore, an essential optimization is to compute only one time each needed block. `QBD_sim_hom` computes only once the six blocks needed to define an infinite and homogeneous QBD. In the finite and homogeneous case, `QBD_sim_hom_finite` computes the three additional needed blocks, namely

[3]One generator is used to determine the successive states of the process and the other is used to generate random times following an exponential distribution.

$C_{-1}, C_0$, and $C_1$ as they are defined in Equation (1.26). In order to simulate an inhomogeneous QBD, we implement the function `QBD_sim_inh`. This one computes the inner blocks of a given level only the first time this level is reached. Then, the blocks are recorded in a data structure for further use.

Similarly with QBDSolver, the Octave functions included in QBD-Simulator are individually offered in order to ease the work of third-party developers that wish to customize the framework.

## 4.4 Outputs display

The measures we are interested in are the stationary probability of every state and the expected first passage times. Their values are always displayed in two forms. First, they are displayed in a text area. At any moment, the user can choose to register the content of this area in a text file. Doing this empties the text area. Several charts are also displayed. Small differences in the display of the results appear depending on the chosen type of analysis and the computation method. We now point out these differences.

When an ordinary performance analysis is carried out, the stationary probability of every level is displayed. When the QBD is infinite, the display is limited to the number of levels previously specified by the user. A chart showing the evolution of the stationary probability according to the level is shown as well (see Figure 4.7).

First passage times are also displayed. There is, however, a significant difference depending on the computation method. More precisely,

- if the matrix analytic methods are chosen, the user can decide to display the expected first passage times starting from any level, but
- if the simulation approaches are used, then only the first passage times from the starting level of the process are displayed.

Furthermore, as we have already discussed in Chapter 3, the results returned by a simulation are merely estimates. Because of that, we recommend to perform more than one replications of the simulation, or to set the number of batches to more than one, enabling the method of batch means. In this case, the tool also computes confidence intervals with a 0.95 confidence level for a chosen measure and it displays this interval in a chart (see Figure 4.8).

When a sensitivity analysis is performed instead of an ordinary analysis, the outputs are displayed differently. For recall, a sensitivity analysis consists in the computation of the performance measures for every perturbed

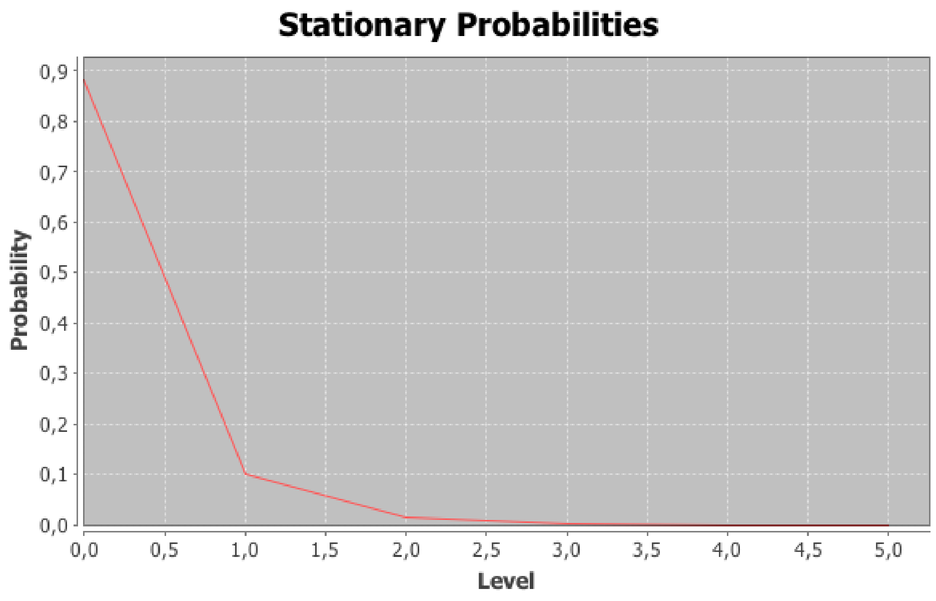


Figure 4.7: The stationary probabilities of the the first six levels of the QBD defined in Example 4.2.1

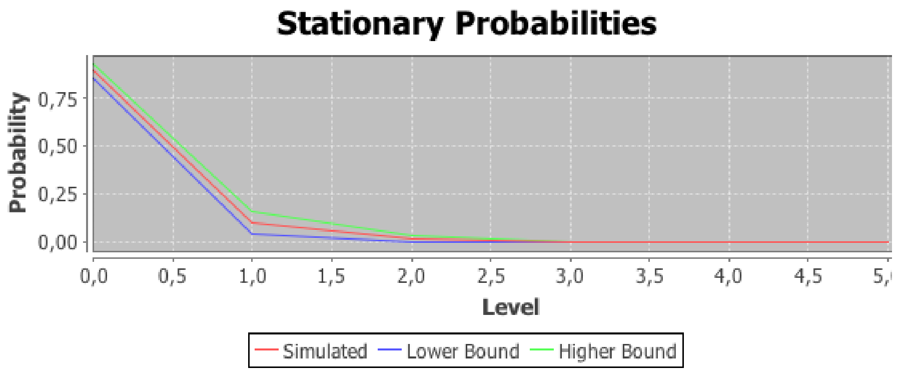


Figure 4.8: The stationary probabilities of the first six levels of Example 4.2.1 estimated by independent replications of a simulation.

generator $\tilde{Q}(\epsilon)$, as defined in Equation (4.14). More precisely, we give different values to $\epsilon$ and we analyse the performance of the resulting perturbed generators. Instead of displaying the whole stationary probability vector, the tool shows the evolution of the stationary probability of a given level in function of $\epsilon$. Similarly, it displays the evolution of the first passage times from a given level to another level according to the value of $\epsilon$. As we previously mentioned, when the simulation is used, the level from which the first passage times are computed can only be the starting level of the process. Furthermore, confidence intervals with a 0.95 confidence level can be computed, provided that multiple replications are performed or that the number of batches set by the user is greater than one.

## 4.5 Architecture of the tool

As previously mentioned, the tool is composed of input and output interfaces as well as the performance analysis frameworks. These individual components are linked together thanks to JAVA components we call **linkers**. In this section, we describe the architecture of the tool and we specifically put emphasis on the linkers. In order to avoid weighing down the reading, the presentation we give remains high level. More accurate information about the architecture can be found in the developer guide provided along with the tool.

Figure 4.9 illustrates the architecture of the tool with a UML component diagram, where the different components and their dependencies are shown. More precisely, a full circle represents a service offered by the component attached to it, and a half circle models a service needed by the component linked to it. In the previous sections, we already presented the two OCTAVE frameworks, as well as QBDPARSER and the user interfaces. Our tool also includes several components developed in JAVA. We briefly describe the role of each of the components. First, it is worth mentioning that we make use of two external JAVA libraries: JFreeChart and JavaOctave.

`JFreeChart`[4] is a free library that allows the developers to display charts inside their JAVA applications. It is distributed under the GNU Lesser General Public License[5]. It supports a large variety of chart types and provides a complete Application Programming Interface (API) for dynamically editing a chart, as well as performing zooms on it.

`JavaOctave` is a JAVA library that works as a bridge from JAVA to OCTAVE. It is developed by Kim Hansen[6]. It allows to call the OCTAVE interpreter from a JAVA program and to transform an OCTAVE data structure into a JAVA object. Basically, it runs OCTAVE and provides functions to communicate with it.

Next, we describe our own JAVA components. `OctaveCaller` is a JAVA component we developed. It is the only component that communicates with `JavaOctave`. Thanks to this property, modularity is improved: the other JAVA components are completely independent from the library used to com-

[4]http://www.jfree.org/jfreechart/
[5]http://www.gnu.org/copyleft/lesser.html
[6]http://kenai.com/projects/javaoctave/

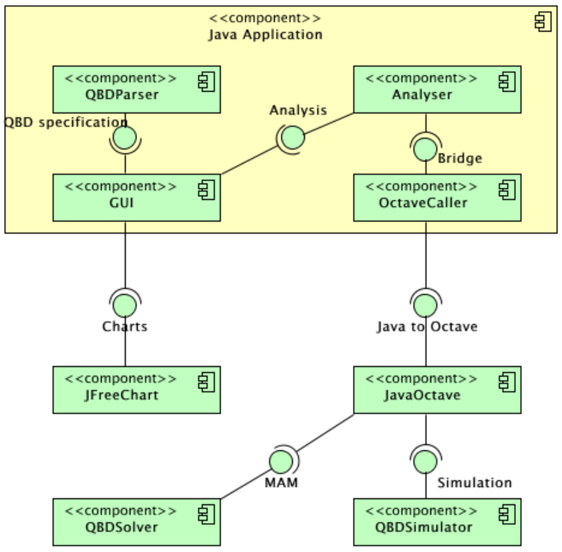


Figure 4.9: The architecture of the tool illustrated with a UML component diagram.

municate with OCTAVE. Since `JavaOctave` is a bridge between JAVA and OCTAVE, `OctaveCaller` can be seen as a gate on the JAVA side. It provides routines to keep track of every OCTAVE function that has been declared. Any function can then be called and its return values will be contained in JAVA objects managed by `OctaveCaller`.

The component `Analyser` is the masterpiece of the tool. It determines which function must be called, depending on the chosen analysis method, the characteristics of the defined QBD process and the desired output data. It determines which function has to be registered by `OctaveCaller`, when and how such a function must be modified. It also sets the right parameters of a function and orders the `OctaveCaller` to call it. Then, it analyses the data structures returned by the `OctaveCaller` to extract the desired results.

Finally, the component `Graphic User Interface` (`GUI`) has two main roles. Firstly, it displays the different windows through which the user can navigate. That includes both the input windows, in which the parameters of the studied system and the path to essential files are entered, as well as the output windows in which the evaluation results are displayed. Secondly, it manages the user actions and requests the right service of the right component when necessary. Basically, this component reads the data introduced by the user, uses the right input interface in order to set every parameter and call `Analyser` to perform the analysis.

## 4.6 Limits and perspectives

QBDSense has two main purposes. First, it aims to facilitate the specification of a QBD by providing different input interfaces. Second, it provides algorithms issued from the matrix analytic methods, as well as some implementations of the discrete-event simulation approaches. However, it suffers from several limitations that we point out in this section.

As previously stated, a Quasi-Birth-and-Death process can be defined using three different methods. Specifying it by coding an Octave function that returns its inner block is the most expressive one. It can be really laborious though, and that is the very reason of the two other methods. However, the expressiveness of these ones is quite limited. Indeed, the Kendall's notation method is restricted to a few queueing systems. Furthermore, it does not allow to customize the configuration of the system, for example by adding some additional servers or by specifying a different queue policy. It is also impossible to define different types of jobs that have their own arrival distribution and a distinct priority.

The last method, *i.e.* giving the possible transitions of the QBD in a text file, is also restricted because of the grammar. Indeed, the language allows to define the transitions of only a two-dimensional markovian state space. Furthermore, the dimensions are explicitly the level and the phase of the corresponding QBD. Therefore, a $n$-dimensional state space must be transformed into a two-dimensional one in order to be specified. To improve our specification method, it is thus required to extend the grammar. However, the use of the method can still be laborious when the process does not clearly have a repetitive structure. In this case, we recommend the use of the first specification method. For example, we encountered this drawback when we carried out the case study presented in Chapter 5.

Other possible improvements of the tool concern the computation of the results. The basic theory of the matrix analytic methods applied to QBDs has been developed more than a decade ago. Researchers still extend these methods. For example, Leeuwaarden *et al.* [21, 20] develop efficient algorithms for some particular classes of QBDs. Cao *et al.* [3], Li *et al.* [22], and Dendievel *et al.* [8] determine expressions that, for a given QBD and a given perturbation, efficiently approximate the resulting performance measures.

Similarly, the implementations of the simulation framework follows only the basic discrete-event simulation approaches. In particular, one key problem of these methods is the bias due to the transient states. Some recent works intend to avoid this bias. For example, Casella *et al.* [4], Dimakos [9], and Thonnes [31] study an approach called perfect simulation. It aims to draw samples from the exact stationary distribution, instead of from a long-time approximation. The integration of perfect simulation algorithms would be a nice addition to our tool.

# Chapter 5

# Case Study

In Chapter 4, we presented QBDSense, a tool that aims to evaluate the performance of a Quasi-Birth-and-Death process. The tool makes intensive use of the algorithms presented in Chapter 2 and the simulation approaches described in Chapter 3. In this chapter, we present a case study we made with the tool. The case study has two purposes. First, it illustrates the use of our tool, from the QBD specification to the presentation of the results. Second, it shows that a QBD can be used to model complex systems. More precisely, the case study is about a reliable system composed of several components and procedures. Montoro and Perez [23] evaluate this system by computing different performance measures. We consider the same system as they presented and we carry out a sensitivity analysis specifically on one of these performance measures.

The chapter is presented as follows. We describe in Section 5.1 the individual components of the system, as well as the procedures and the random events that have an impact on it. Next, we develop a mathematical model of the system and we formally define the state space of the model in Section 5.2. In particular, we give the probability distributions that characterize the random events. In Section 5.3, we present the method used afterwards to define the QBD modelling the system. The definition of the generator of the QBD is given in Section 5.4. Finally, we define the performance measure in which we are interested and we perform the numerical evaluation in Section 5.5.

## 5.1 System description

In this section, we describe the system. We present its architecture in Figure 5.1. The system is composed of identical working units such that:

- among these units, at most one is online and the others are either in warm standby or in repair;

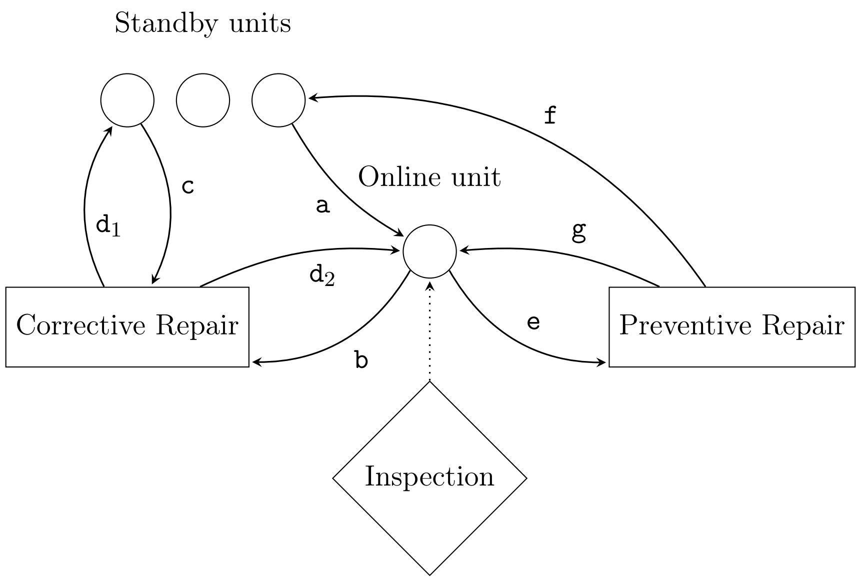


Figure 5.1: A repairable system under degradation, inspection, and two types of repair.

- all the units have a finite lifespan, which can be regarded as a set of degrading phases; this means that both the online unit and the standby units degrade over time until they eventually break;
- every phase of the online unit is labelled either “good” or “bad”
- if the online unit breaks, then a standby unit, if there is any, immediately replaces it and becomes the online one (transition labelled `a` in Figure 5.1). The replacing unit is considered as good as new when it becomes online.

The global system is considered online as long as there is an online unit.

Once a unit breaks, it enters the corrective repair procedure (transitions labelled `b` and `c`). Only one unit can be repaired at the same time. The other ones wait in a queue. Once the unit being repaired is newly working, the first broken unit in the queue, if any, starts to be repaired. A newly repaired unit is as good as new. If there is already an online unit, the repaired unit returns in standby as soon as the repair is completed (transition labelled $\mathtt{d}_1$). Otherwise, it becomes the online unit (transition labelled $\mathtt{d}_2$).

Occasionally, a inspection procedure is carried out. It consists in the analysis of the current degrading phase of the online unit. If this phase is labelled "good", nothing happens. Otherwise, the online unit enters a preventive repair procedure (transition labelled `e`). Similarly with the corrective repair procedure, only one unit can be preventively repaired at a time. The other units that must be preventively repaired wait in a queue. When the repair is completed, the unit goes back in standby and is as good as new (transition labelled `f`). Finally, if no unit is available to work, that is they are all either in preventive or corrective repair, then the unit currently being preventively repaired, if any, urgently becomes the online unit (transition labelled `g`).

## 5.2 Mathematical model

In this section, we give the mathematical model of the system such as it is defined by Montoro and Perez [23]. First, we define the following parameters:

- $n \geq 1$ is the total number of units,
- $m > 1$ is the number of degrading phases of the online unit before it is broken, and
- $g : 1 \leq g \leq m-1$ is the number of these degrading phases that are labelled "good".

In other words, among the $m$ degrading phases of the online unit, the first $g$ states are labelled "good" and the other $m-g$ states are labelled "bad".

Next, we feature the random events that have an impact on the system state. More precisely,

- the lifespan of the online unit follows a $PH(\alpha, T)$ distribution (see Definition 1.2.5) with $m$ phases, each phase corresponding to a degrading phase
- the lifespan of a standby unit follows a $PH(\alpha_s, T_s)$ distribution with $m_s$ phases, with $m_s \geq 1$,
- the corrective repair time follows a $PH(\beta_c, S_c)$ distribution with $n_c$ phases, with $n_c \geq 1$,
- the preventive repair time follows a $PH(\beta_p, S_p)$ distribution with $n_p$ phases, with $n_p \geq 1$, and
- the time between two inspections follows a $PH(\gamma, L)$ distribution with $v$ phases, with $v \geq 1$.

All of these random times are considered to be independent of each other. Finally, if a unit being preventively repaired must urgently becomes the online unit, its new degrading phase is determined with a $m$ sized probability vector, denoted by $\alpha_p$.

According to these definitions, Montoro and Perez [23] model the state of the system with a vector

$$(c, p, d, z_1, ..., z_{n-1-(i+j)}, r_c, r_p, f)$$

where

- $c \in \{0, \ldots, n\}$ is the number of units in corrective repair,
- $p \in \{0, \ldots, n-1\}$ is the number of units in preventive repair,
- $d \in \{1, \ldots, m\}$ is the degrading phase of the online unit,
- for $h : 1 \leq h \leq n-1-(c+p)$, $z_h \in \{1, \ldots, m_s\}$ is the lifespan phase of the $h^{\text{th}}$ standby unit,
- $r_c \in \{1, \ldots, n_c\}$ is the phase of the currently performed corrective repair,
- $r_p \in \{1, \ldots, n_p\}$ is the phase of the currently performed preventive repair, and
- $f \in \{1, \ldots, v\}$ is the inspection phase.

After defining the state space of the system, we model it with a Quasi-Birth-and-Death process. In order to carry out a numerical evaluation, we must use one of the input interfaces presented in Chapter 4 to specify the QBD. Clearly, the input interface based on Kendall's notation is not appropriate because the system is far more complex than a single-server queueing system. Furthermore, as mentioned in Section 4.6, the grammar we developed to specify the transitions of a QBD is laboriously applicable when the state space of the studied process has more than two dimensions. Hence, we propose to build up the generator by explicitly giving the content of the inner blocks.

## 5.3 An approach to model a composite system

As previously observed, the system can be regarded as the composition of individual subsystems. We call this kind of system "composite". Subsequently, the state of the system depends on many random events. Because of that, the definition of the infinitesimal generator of the QBD modelling such a system is rather complex. Therefore, we propose first to give the guidelines of the approach we follow to model this composite system with a QBD.

First, we present mathematical operators that can be used to more concisely specify the inner blocks of the QBD generator in Subsection 5.3.1. Next, we define a simpler version of the repairable system in Subsection 5.3.2. Through all the section, we make use of this simpler system to illustrate the principles of our approach. We model this simpler system with a QBD and build a specific inner blocks composing the QBD generator in Subsection 5.3.3.

### 5.3.1 Kronecker product and sum

Matrices such as the inner blocks we built afterwards can be more concisely specify by using particular matrix operators. In this subsection, we give a definition of these operators, according to Horn *et al.* [14].

**Definition 5.3.1 (Kronecker product)**
*Let $A$ be an $m \times n$ matrix and $B$ be a $p \times q$ matrix. Then, the* ***Kronecker product*** *of $A$ and $B$, which we denoted by $A \otimes B$, is the $mp \times nq$ matrix defined as*

$$A \otimes B = \begin{pmatrix} a_{11}B & \dots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \dots & a_{mn}B \end{pmatrix} \tag{5.1}$$

The definition of Kronecker product allows us to define another matrix operator that we will use intensively afterwards.

**Definition 5.3.2 (Kronecker sum)**
*Let $A$ be an $m \times m$ matrix and $B$ be a $p \times p$ matrix. Then, the* ***Kronecker sum*** *of $A$ and $B$, which is denoted by $A \oplus B$, is the $mp \times mp$ matrix defined as*

$$A \oplus B = A \otimes I_p + I_m \otimes B \tag{5.2}$$

*where $I_k$ denotes the $k \times k$ identity matrix.*

### 5.3.2 Definition of a simple repairable system

We consider the repairable system described in Section 5.1 with only two units but without any inspection procedure or preventive repair. Subsequently, among the transitions shown in Figure 5.1, only the ones labelled **a**, **b**, **c**, $\mathbf{d}_1$, and $\mathbf{d}_2$ are available. We define that

1. the lifespan of the online unit is $PH(\alpha, T)$ distributed where $\alpha$ is a $1 \times 2$ row vector and $T$ is a $2 \times 2$ matrix,
2. the lifespan of the standby unit is $PH(\alpha_s, T_s)$ distributed where $\alpha_s$ is a $1 \times 2$ row vector and $T_s$ is a $2 \times 2$ matrix, and
3. the corrective repair time is $PH(\beta_c, S_c)$ distributed, where $\beta_c$ is a $1 \times 2$ row vector and $S_c$ is a $2 \times 2$ matrix.

Subsequently, we can model the state of the system with a vector

$$(c, d, z)$$

where

- $c \in \{0, 1, 2\}$ is the number of units in corrective repair,
- $d \in \{1, 2\}$ is the degrading phase of the online unit,
- $z \in \{1, 2\}$ is either the lifespan phase of the unit in standby (when $c = 0$) or the phase of the currently performed corrective repair (when $c > 0$).

according to the mathematical model we defined in Section 5.2.

### 5.3.3 Modelling the simple repairable system

In this subsection, we use a QBD to model this system. The level of the QBD is $c$, that is the number of units in corrective repair. The phase is determined according to $d$ and $z$. When in level 0, there is no unit in corrective repair. Accordingly, the phase of the QBD depends only on the degrading phase of the online unit and the lifespan phase of the standby unit.

Because the corresponding phase-type distributions have only two phases, there are only four states in level 0, namely

$$l(0) = \{(0,1,1), (0,1,2), (0,2,1), (0,2,2)\}. \tag{5.3}$$

When in level 1, there is no more a standby unit, but a corrective repair is performed. In this level, the phase of the QBD is determined according to the degrading phase of the online unit and the corrective repair phase. Similarly with level 0, there are only four states in level 1, namely

$$l(1) = \{(1,1,\underline{1}), (1,1,\underline{2}), (1,2,\underline{1}), (1,2,\underline{2})\}. \tag{5.4}$$

It is noteworthy that we underlined the phase of the corrective repair to distinguish it from the phase of the standby unit.

We are first interested in characterizing the transition rates from a state of level 0 to a state of level 1. For example, let us consider the transition

$$(0, 1, 1) \longrightarrow (1, 1, \underline{1})$$

which is the one corresponding to the breaking of the online unit given that

1. the online unit was in degrading phase 1,
2. the standby unit becomes online and enters phase 1,
3. the standby unit was in lifespan phase 1, and
4. the corrective repair of the broken unit starts in phase $\underline{1}$.

It is noteworthy that

1. the rate of probability that the online unit breaks given that it was in phase 1 is $t_1$,
2. the probability that the new online unit starts in phase 1 is $\alpha_1$,
3. the phase of the standby unit before the breaking has no impact on the transition, and
4. the corrective repair starts in phase $\underline{1}$ with probability $(\beta_c)_1$.

Accordingly, the transition occurs with a rate of

$$t_1 \alpha_1 1 (\beta_c)_1. \tag{5.5}$$

It is worth mentioning that although we could have removed it, the scalar 1 in the product has a special meaning. Indeed, it means that the phase of the standby unit has no consequence on the transition rate.

We use a similar approach to determine the rate of the other transitions from level 0 to level 1. These ones are given in Table 5.1. According to these transition rates, we observe that the inner block recording them, that is $B_1$ (see Equation (1.21)), can be concisely written in a concise form as

$$B_1 = t\alpha \otimes e_2 \otimes \beta_c \tag{5.6}$$

where $e_2$ is a column vector of size 2 filled with ones.

As already illustrated with the simple repairable system, the approach we follow is perfectly convenient for specifying the infinitesimal generator of the complete repairable system we described in Section 5.1. However, because the complete system is composed of more subsystems, determining the form of its generator is far more laborious. In the next section, we subsequently propose to focus on two of the inner blocks.

| From \ To | $(0,1,\underline{1})$ | $(0,1,\underline{2})$ | $(0,2,\underline{1})$ | $(0,2,\underline{2})$ |
|---|---|---|---|---|
| $(0,1,1)$ | $t_1\alpha_1(\beta_c)_1$ | $t_1\alpha_1(\beta_c)_2$ | $t_1\alpha_2(\beta_c)_1$ | $t_1\alpha_2(\beta_c)_2$ |
| $(0,1,2)$ | $t_1\alpha_1(\beta_c)_1$ | $t_1\alpha_1(\beta_c)_2$ | $t_1\alpha_2(\beta_c)_1$ | $t_1\alpha_2(\beta_c)_2$ |
| $(0,2,1)$ | $t_2\alpha_1(\beta_c)_1$ | $t_2\alpha_1(\beta_c)_2$ | $t_2\alpha_2(\beta_c)_1$ | $t_2\alpha_2(\beta_c)_2$ |
| $(0,2,2)$ | $t_2\alpha_1(\beta_c)_1$ | $t_2\alpha_1(\beta_c)_2$ | $t_2\alpha_2(\beta_c)_1$ | $t_2\alpha_2(\beta_c)_2$ |

Table 5.1: The transition rates from level 0 to level 1 of the simple repairable system.

## 5.4 Specification of the infinitesimal generator

In this section, we build the inner blocks composing the QBD generator and define the level of the QBD as the number of units that are non-working, namely the ones either in corrective repair or in preventive repair. More formally, we define level $k$, with $0 \leq k \leq n$, as

$$l(k) = \{(c, p, d, z_1, ..., z_{n-1-(i+j)}, r_p, r_c, f) : c + p = k\} \tag{5.7}$$

As already mentioned, the building of the generator is laborious. We choose to focus on some of the inner blocks. Our intent is to put emphasis on the general principles of the approach followed to build the generator. We specifically apply this approach to compute inner blocks $B_1$ and $B_0$.

### 5.4.1 Transitions from level 0 to level 1

First, we must put emphasis on the subsystems that run in each of the considered levels. When in level 0, the only running subsystems are

- the online unit,
- the $n-1$ standby units, and
- the inspection procedure.

Subsequently, a state of level 0 has the form

$$(0, 0, d^{(0)}, z_1^{(0)}, \ldots, z_{n-1}^{(0)}, f^{(0)}).$$

In level 1, however, the running subsystems are

- the online unit,
- $n-2$ standby units,
- either a corrective repair or a preventive repair, and
- the inspection procedure.

We propose to divide the state space of level 1 into two sets as follows.

1. The first set includes the states corresponding to a unit in corrective repair. Therefore, a state of this set has the form

$$(1, 0, d^{(1)}, z_1^{(1)}, \dots, z_{n-2}^{(1)}, r_c, f^{(1)}).$$

   We record the transitions that lead to such a state in a matrix denoted by $B_1(1)$.

2. The second set includes the states corresponding to a unit in preventive repair. A state of this set has the form

$$(0, 1, d^{(1)}, z_1^{(1)}, \dots, z_{n-2}^{(1)}, r_p, f^{(1)}).$$

   We record the transitions that reach such a state in a matrix denoted by $B_1(2)$.

According to this division of the state space, inner block $B_1$ is given by

$$B_1 = \begin{pmatrix} B_1(1), & B_1(2) \end{pmatrix} \tag{5.8}$$

We characterize the transition rates of each of the two matrices composing $B_1$. We focus first on the transition

$$(0, 0, d^{(0)}, z_1^{(0)}, \dots, z_{n-1}^{(0)}, f^{(0)}) \longrightarrow (1, 0, d^{(1)}, z_1^{(1)}, \dots, z_{n-2}^{(1)}, r_c, f^{(1)}).$$

Only two types of event result in such a transition. These are

1. either the online unit breaks and goes to corrective repair in phase $r_c$, given that it was in degrading phase $d^{(0)}$, and accordingly the first standby unit becomes online, starting in phase $d^{(1)}$;

2. or one of the standby units breaks and goes to corrective repair in phase $r_c$, given that it was in lifespan phase $z_i^{(0)}$, $i : 1 \le i \le n-1$ being the number of the broken standby unit.

No other transition may happen (for example, a modification of the phase of the inspection procedure) and these events are independent. Therefore, the resulting rate of the transition is the sum of the rate of occurrence of each of these events.

Let us consider the first event. It is noteworthy that:

1. The rate corresponding to the breaking of the online unit given that it was in phase $d^{(0)}$ is $t_{d^{(0)}}$.

2. A standby unit immediately replaces the online unit and its initial degrading phase is $d^{(1)}$ with probability $\alpha_{(d^{(1)})}$.

3. The lifespan phase of the replacing standby unit has no influence on the occurrence rate of the event.
4. The lifespan phase of the other standby units is not modified.
5. The initial phase of the corrective repair is $r_c$ with $(\beta_c)_{r_c}$.
6. The phase of the inspection procedure is not modified.

Subsequently, the rate of occurrence of the first event is given by

$$t_{d^{(0)}}\ \alpha_{(d^{(1)})}\ 1[z_1^{(0)}]\ 1[z_2^{(0)} = z_2^{(1)}]\ \ldots\ 1[z_{n-1}^{(0)} = z_{n-1}^{(1)}]\ (\beta_c)_{r_c}\ 1[f^{(0)} = f^{(1)}]$$

which is equivalent to

$$t_{d^{(0)}}\alpha_{(d^{(1)})}(\beta_c)_{r_c}.$$

Next, we determine the rate of the second event according to a similar approach. We assume that the broken standby unit is the ith one. We observe that:

1. The degrading phase of the online unit is not modified.
2. The breaking rate of the ith standby unit given that it was in phase $z_i^{(0)}$ is $(t_s)_{z_i^{(0)}}$ where
$$t_s = -T_s\mathbf{1}. \tag{5.9}$$
3. The lifespan phase of the other standby units is not modified.
4. The initial phase of the corrective repair is $r_c$ with $(\beta_c)_{r_c}$.
5. The phase of the inspection procedure is not modified.

Hence, the occurrence rate of the second event is given by

$$\begin{gathered}1[d^{(0)} = d^{(1)}]\ 1[z_1^{(0)} = z_1^{(1)}]\ \ldots\ 1[z_{i-1}^{(0)} = z_{i-1}^{(1)}]\ (t_s)_{z_i^{(0)}}\\ 1[z_{i+1}^{(0)} = z_{i+1}^{(1)}]\ \ldots\ 1[z_{n-1}^{(0)} = z_{n-1}^{(1)}]\ (\beta_c)_{r_c}\ 1[f^{(0)} = f^{(1)}]\end{gathered}$$

which is equivalent to

$$(t_s)_{z_i^{(0)}}(\beta_c)_{r_c}.$$

To summarize, the total rate of the transition is

$$t_{d^{(0)}}\alpha_{(d^{(1)})}(\beta_c)_{r_c} + (t_s)_{z_i^{(0)}}(\beta_c)_{r_c}.$$

As in Subsection 5.3.3, we concisely write matrix $B_1(1)$ in algebraic form

$$\begin{aligned}B_1(1) = t\alpha \otimes e_{m_s} \otimes I_{(m_s)^{n-2}} \otimes \beta_c \otimes I_v\\ + I_m \otimes (t_s \oplus \cdots \oplus t_s)_{n-1} \otimes \beta_c \otimes I_v\end{aligned} \tag{5.10}$$

It is noteworthy that we used a vector filled with ones, namely $e_{m_s}$, to express that the phase of the replacing unit has no impact on the transition rate, and that this specific phase must not be recorded anymore after the transition.

We still have to define the second matrix that composes inner block $B_1$. This matrix, $B_1(2)$, records the transitions related to the beginning of the preventive repair of the online unit. For example, we consider the transition

$$(0, 0, d^{(0)}, z_1^{(0)}, \dots, z_{n-1}^{(0)}, f^{(0)}) \longrightarrow (0, 1, d^{(1)}, z_1^{(1)}, \dots, z_{n-2}^{(1)}, r_p, f^{(1)}).$$

This transition is triggered when the online unit is in a state labelled "bad", **and** an inspection occurs. Then, the unit enters the preventive repair and the inspection procedure starts again. More precisely,

1. Such a transition can only occur when the online unit is in one of the last $m - g$ degrading phases, that is
$$d^{(0)} \in \{g + 1, \dots, m\}. \tag{5.11}$$
2. The first standby unit replaces the online unit and its initial phase is $d^{(1)}$ with probability $\alpha_{d^{(1)}}$.
3. The lifespan phase of the replacing unit has no influence on the transition rate.
4. The lifespan phase of the $n - 2$ other standby units has no influence on the transition rate and it is not modified by the transition.
5. The initial phase of the preventive repair is $r_p$ with probability $(\beta_p)_{r_p}$.
6. Given that the inspection procedure is in phase $f^{(0)}$, the rate of occurrence of the inspection is $l_{f^{(0)}}$ where
$$l = -L\mathbf{1}. \tag{5.12}$$
7. The initial phase of the new inspection procedure is $f^{(1)}$ with probability $\gamma_{f^{(1)}}$.

When the degrading phase if the online unit is labelled "bad", the transition occurs with a rate of

$$1[d^{(0)}]\ \alpha_{(d^{(1)})}\ 1[z_1^{(0)}]\ 1[z_2^{(0)} = z_2^{(1)}]\ \dots\ 1[z_{n-1}^{(0)} = z_{n-1}^{(1)}]\ (\beta_p)_{r_p}\ l_{f^{(0)}}\ \gamma_{f^{(1)}}$$

Subsequently, the rate of the transition is given by

$$\begin{array}{rl} 0, & d^{(0)} \in \{1, \dots, g\} \\ \alpha_{(d^{(1)})}\ (\beta_p)_{r_p}\ l_{f^{(0)}}\ \gamma_{f^{(1)}}, & d^{(0)} \in \{1, \dots, g + 1, m\} \end{array}$$

and $B_1(2)$ can be written in algebraic form as

$$B_1(2) = U_2 e_m \alpha \otimes e_{m_s} \otimes I_{m_s^{(n-2)}} \otimes \beta_p \otimes l\gamma \tag{5.13}$$

with

$$U_2 = \begin{pmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & I_{m-g} \end{pmatrix} \tag{5.14}$$

It worth mentioning that matrix $U_2$ is used instead of an identity matrix because the transition is available only when the online unit is in a bad phase. Furthermore, we use a column vector because after the transition, the degrading phase of the broken online unit does not needed to be recorded anymore.

### 5.4.2 Transitions from level 0 to level 0

We build next inner block $B_0$, that is the one recording the intra-level transitions of level 0. For recall, when in level 0, the only running subsystems are

- the online unit,
- the $n-1$ standby units, and
- the inspection procedure.

Therefore, a transition between two states of level 0 is the result of one of the following events:

1. The degrading phase of the online unit is modified but the unit is still operational (otherwise, the process would move to level 1).
2. The lifespan phase of a standby unit is modified but the considered unit is still operational.
3. The inspection procedure reaches another phase but it does not enter the absorbing phase, that is the inspection does not occur yet.
4. The inspection occurs, but the current phase of the online unit is labelled "good".

It is noteworthy that at most one instance of these events can occur at a time. A transition implied by the first event has the form

$$(0, 0, d^{(0)}, z_1, \dots, z_{n-1}, f) \longrightarrow (0, 0, d^{(1)}, z_1, \dots, z_{n-1}, f).$$

Indeed, the first event modifies the phase of the online unit but the phase of the other subsystems is unchanged. Furthermore, the phase of the other subsystems have no influence on the transition rate. More precisely, only the current and the next phase of the online unit determine the rate. Subsequently, the transition rate is $T_{d^{(0)}d^{(1)}}$, given that the online unit moves from phase $d^{(0)}$ to phase $d^{(1)}$.

Similarly, if the phase of the i^th^ standby unit is modified, the phase of the other subsystems does not change. It therefore implies a transition of the form

$$\begin{aligned} &(0,0,d,z_1,\ldots,z_{i-1},z_i^{(0)},z_{i+1},\ldots,z_{n-1},f) \\ \longrightarrow\ &(0,0,d,z_1,\ldots,z_{i-1},z_i^{(1)},z_{i+1},\ldots,z_{n-1},f) \end{aligned}$$

with a rate of $(T_s)_{z_i^{(0)},z_i^{(1)}}$, given that the considered online unit moves from phase $z_i^{(0)}$ from $z_i^{(1)}$.

The third event is also very similar. This time, it is the phase of the inspection procedure that is modified. It implies a transition of the form

$$(0,0,d,z_1,\ldots,z_{n-1},f^{(0)}) \longrightarrow (0,0,d,z_1,\ldots,z_{n-1},f^{(1)})$$

whose rate is determined by $L_{f^{(0)},f^{(1)}}$, given that the inspection procedures moves from phase $f^{(0)}$ to phase $f^{(1)}$.

The fourth event is a bit more particular. It implies a transition of the form

$$(0,0,d,z_1,\ldots,z_{n-1},f^{(0)}) \longrightarrow (0,0,d,z_1,\ldots,z_{n-1},f^{(1)})$$

but unlike the third event, the inspection really occurs. It is noteworthy that

1. such a transition can only occur if the degrading phase of the online unit is in $\{1,\ldots,g\}$ (otherwise, the unit would enter the preventive repair)
2. given that it is in phase $l_{f^{(0)}}$ the inspection enters its absorbing phase with a rate of $l_{f^{(0)}}$;
3. the inspection must be restart and its initial phase if $f^{(1)}$ with probability $(\beta_p)_{f^{(1)}}$;
4. the other subsystems are unchanged and their respective phase has no impact on the transition rate.

Subsequently, this type of transition occurs with a rate of

$$\begin{aligned} l_{f^{(0)}}(\beta_p)_{f^{(1)}}, \quad & d\in\{1,\ldots,g+1\}, \\ 0, \quad & d\in\{g+1,\ldots,m\}. \end{aligned} \tag{5.15}$$

We have then characterized all the intra-level transitions of level 0. We still have to determine the diagonal elements of $B_0$. Indeed, because of the definition of infinitesimal generator (see Equations 1.14), the ith diagonal element of $B_0$ is determined as

$$(B_0)_{ii} = -(\sum_{j \neq i}(B_0)_{ij} + \sum_k (B_1)_{ik}). \tag{5.16}$$

As we did for $B_1$, we can show that the algebraic form of $B_0$ can be written as

$$\begin{aligned} B_0 = &(T \oplus (T_s \oplus \cdots \oplus T_s)_{n-1}) \oplus L \\ &+ U_1 \otimes I_{(m_s)^{n-1}} \otimes l\gamma. \end{aligned} \tag{5.17}$$

where $U_1$ is a $m \times m$ matrix of the form

$$U_1 = \begin{pmatrix} I_g & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix} \tag{5.18}$$

It is noteworthy that we use $U_1$ instead of an identity matrix because the fourth event can only occur when the online unit is in a “good”degrading phase.

Although they are many more blocks to specify, we do not deal with them and we refer to Montoro and Perez [23] for the complete specification of the generator. Their article is given in Appendix C. The purpose of this section and the previous one was to give the keys of our approach towards the specification of such a complex composite system.

In the next section, we give values to the parameters of the system and we perform a numerical evaluation of its performance.

## 5.5 Numerical evaluation

After defining the generator of the Quasi-Birth-and-Death process, we set all the parameters of the system in order to carry out a numerical evaluation. More precisely, the performance measure we are interested in is the rate of occurrence of failure (ROCOF) of the system. In simple terms, the ROCOF can be seen as the probability that the system fails in a small interval of time. The goal of our case study is to perform a sensitivity analysis in order to determine the influence of

1. the rate of occurrence of inspections, and
2. the labelling policy of the degrading phases of the online unit

on the ROCOF of the system.

For recall, the system fails when there are $n-1$ units in corrective repair and the online unit breaks. Indeed, in this case, there is no unit in standby or in preventive repair. Therefore, the broken online unit cannot be replaced.

According to Montoro and Perez [23], the ROCOF of the system is given by

$$r = \pi_{n-1} \begin{pmatrix} t \otimes e_{n_c v} \\ \mathbf{0} \end{pmatrix}. \tag{5.19}$$

Indeed, according to our mathematical model, if there are $n-1$ units in corrective repair, then the QBD is in level $n-1$. The stationary probabilities of the level $n-1$ states are recorded in $\pi_{n-1}$. The online unit breaking rates are recorded in vector $t$. The current corrective repair as well as the inspection procedure has no influence on the breaking (hence the column vector of size $n_c v$ filled with ones). Finally, a vector filled with zeros is used because the system cannot fail if there is at least one unit in preventive repair. Otherwise, the unit being preventively repaired would immediately replace the broken online unit.

We choose to keep the parameters values as they are defined by Montoro and Perez [23] in their own case study. First, the lifespan of the online unit is $PH(\alpha, T)$ distributed where

$$\alpha = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{5.20}$$

$$T = \begin{pmatrix} -0.0081 & 0.0081 & 0 & 0 & 0 \\ 0 & -0.024 & 0.024 & 0 & 0 \\ 0 & 0 & -0.009 & 0.009 & 0 \\ 0 & 0 & 0 & -0.0072 & 0.0072 \\ 0 & 0 & 0 & 0 & -0.84 \end{pmatrix} \tag{5.21}$$

Accordingly, the lifespan of the online unit can be regarded as an increasing degradation that eventually lead to a breaking. This breaking can only occur when the online unit is in its most critical phase, namely the last one. It is also important to note that a new online unit always starts in the less critical phase, that is the first one.

The lifespan of a standby unit is $PH(\alpha_s, T_s)$ distributed where

$$\alpha_s = \begin{pmatrix} 1 & 0 \end{pmatrix}, \tag{5.22}$$

$$T_s = \begin{pmatrix} -0.0052 & 0.0052 \\ 0.0013 & -0.0052 \end{pmatrix}. \tag{5.23}$$

Unlike an online unit, a standby unit can move to its most critical phase (*i.e.* the last phase) back to its safest phase (namely, the first phase). Furthermore, a standby unit always starts in the safest phase.

Next, we define the distribution followed by the repair times. The corrective repair time is $PH(\beta_p, T_s)$ distributed where

$$\beta_c = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \tag{5.24}$$

$$S_c = \begin{pmatrix} -0.02 & 0.02 & 0 \\ 0.01 & -0.08 & 0.07 \\ 0.005 & 0 & -0.10 \end{pmatrix} \tag{5.25}$$

and the preventive repair time is $PH(\beta_c, T_s)$ distributed where

$$\beta_p = \begin{pmatrix} 1 & 0 \end{pmatrix}, \tag{5.26}$$

$$S_c = \begin{pmatrix} -0.0667 & 0.0667 \\ 0 & -0.0667 \end{pmatrix}. \tag{5.27}$$

As stated above, the unit in preventive repair can be required to become the online unit. It happens when the online unit breaks and when there are no unit in standby. In this case, the initial degrading phase of the unit is given by

$$\alpha_p = \begin{pmatrix} 0.1 & 0.3 & 0.2 & 0.3 & 0.1 \end{pmatrix} \tag{5.28}$$

Finally, the inter-inspection times are $PH(\gamma, L)$ distributed where

$$\gamma = \begin{pmatrix} 1 & 0 \end{pmatrix}, \tag{5.29}$$

$$L = \begin{pmatrix} -0.02 & 0.02 \\ 0 & -0.02 \end{pmatrix}. \tag{5.30}$$

Then, we perform a sensitivity analysis of the ROCOF of the system. More precisely, we define a perturbation on the inspection distribution:

$$\tilde{L}(\epsilon) = L + \epsilon A \tag{5.31}$$

with

$$A = \begin{pmatrix} -1 & 1 \\ 0 & -1 \end{pmatrix} \tag{5.32}$$

and where $\epsilon$ will successively receive different values. It is noteworthy that the greater the $\epsilon$ value, the shorter the time between two inspections. Reducing the inter-inspections time is likely to increase the rate at which an online unit enters the preventive repair.

According to these definitions and the form of the infinitesimal generator of the QBD, we implemented an OCTAVE function that returns the three

| $g/\ \epsilon$ | 0.01 | 0.03 | 0.05 | 0.07 | 0.09 |
|---|---|---|---|---|---|
| 1 | 1.4921e-06 | 8.3196e-07 | 6.3283e-07 | 5.4581e-07 | 4.9886e-07 |
| 2 | 2.4186e-06 | 1.5270e-06 | 1.2121e-06 | 1.0623e-06 | 9.7770e-07 |
| 3 | 5.9177e-06 | 4.4985e-06 | 3.8340e-06 | 3.4591e-06 | 3.2224e-06 |
| 4 | 1.5043e-05 | 1.4081e-05 | 1.3302e-05 | 1.2666e-05 | 1.2141e-05 |

Table 5.2: Evolution of the ROCOF of the repairable system (table).

inner blocks related to a level given as parameters. We have also included $\epsilon$ as a second parameter of the function, as it has been explained in Section 4.2.4.

We also want to assign different values to parameter $g$, *i.e.* the number of degrading phases of the online unit that are labelled "good". Because of the definition of this parameter, the online unit is more likely to be preventively repaired for a lower value of $g$. However, our tool does not allow to perform a sensitivity analysis with two variation parameters. Furthermore, if we use the tool via the graphical user interface we developed, the only interesting results that would be obtained are the values contained in $\pi_{n-1}$. Therefore, we propose to profit from the modularity of our tool and to use the framework `QBDSolver` separately (see Subsection 4.3.1). More precisely, we implement an OCTAVE function especially for the case study. For each value of $g$ and $\epsilon$, this function calls the functions provided by the framework to compute $\pi_{n-1}$. Then, it computes the ROCOF of the system according to Equation (5.19).

We give in Table 5.2 the results of our sensitivity analysis. A chart showing these results is also given in Figure 5.2. It is clear from these results that either reducing the number of good degrading phases or increasing the inspection rate helps to decrease the ROCOF of the system. In other words, sending the online unit in preventive repair more rapidly has a good benefit on the system robustness. Thanks to these results, a compromise can be made between the inspection rate and the considerations with regard to the degrading phases of the online unit in order to decrease the ROCOF of the system.

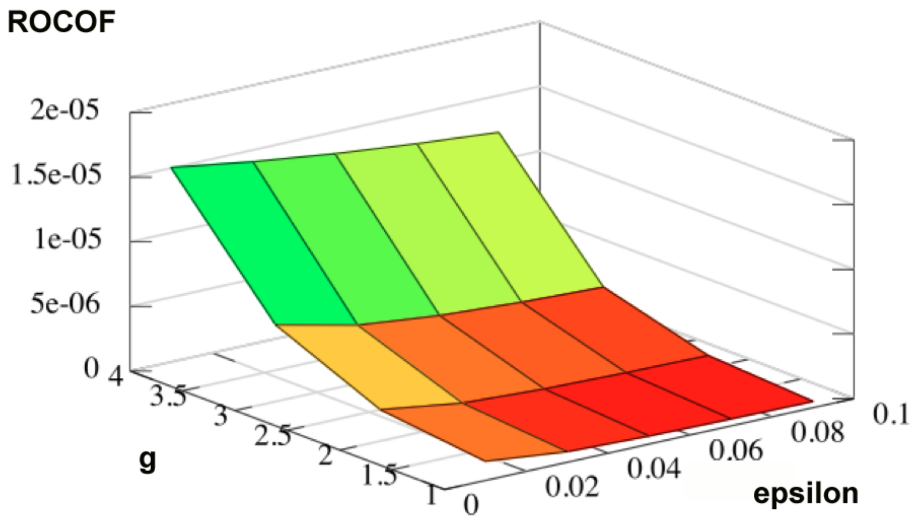


Figure 5.2: Evolution of the ROCOF of the repairable system (chart).

# Part IV

# Conclusion

Chapter 6

# Review and perspectives

## 6.1 Work Summary

Through this thesis, we introduced the Quasi-Birth-and-Death process as a convenient model for analysing random systems and we presented formal methods to carry out a performance analysis. As we stated in the introduction, we focused mainly on three steps of the performance analysis procedure, such as it is described by Leemis and Park [19].

First, we dealt with the definition of a **specification model** based on a QBD. This part of the analysis is spread across several chapters. In Chapter 1, we gave a formal definition of the QBD process. In Chapter 4 , we described our tool, which includes three input interfaces to specify a QBD. The purpose of this specification is the building of the matrix recording the possible transitions of the process. Finally, a large part of the case study presented in Chapter 4 described an informal yet convenient approach towards the definition of a QBD modelling a composite system.

Second, a **computational model**, namely a computer program that computes performance measures, can then be built thanks to the algorithms presented in Chapter 2 and in Chapter 3. Both chapters present a particular type of methods to compute the measures. The former is related to explicit computation algorithms, which make use of costly matrix operations. To overcome this drawback, the latter chapter presents a general algorithm to simulate a QBD and methods to estimate the performance measures. The specification of a given QBD and the use of any type of computation algorithm somehow form the so-called computational model.

Finally, the **verification** of the computational model is eased because of the strong coupling between the specification model and the computational model inherent in our analysis method. Indeed, provided that there are no human errors, our input interfaces guarantee that the data structure recorded in the tool is exactly the QBD used to model the system. Furthermore, the computation algorithms are based on proven mathematical theorems. The verification step thus consists in testing that the implementations are consistent with the algorithms.

## 6.2 Contributions of the thesis

Although they do not always appear explicitly, the contributions of this work are worth mentioning.

First, the thesis is supposed to be readable by any computer scientist that does not have a deep knowledge in mathematics, especially in stochastic modelling. Indeed, we recalled the basic definitions of the probability theory and the stochastic processes in Chapter 1. Through all the chapters, we aimed to combine rigour with intuitive reasoning. In particular, we aimed to present the matrix analytic methods without making the reader overwhelmed by mathematical equations, theorems, or formal proves.

Second, we gave a synthesis of the analytical studies related to the QBDs. We also applied the discrete-event simulation approaches specifically to Quasi-Birth-and-Death processes in Chapter 3. One of these two kinds of computation method can therefore be used during an analysis.

One of our main contributions is our particular concern for the specification of a QBD. While there exist tools, such as MAMSolver [28], that allow to give the parameters of some queueing systems with a text file containing numbers, none of them provides a user-friendly method to concisely specify a QBD. We applied the principles of the language theory to develop our own QBD specification language. In spite of its limitations, which we already mentioned in Chapter 4, our language constitutes an essential part of our work.

Finally, we developed the QBDSense tool. This tool integrates three input interfaces (included our language) and allows to perform two types of analysis using two computation methods. We also tried to make it modular enough so that third-party developers can either modify it or extract some parts of it for their own tool. In particular, the computations methods are provided as a set of Octave functions that can be used independently of the tool. Furthermore, modifying one of the computation algorithms is transparent with regard to the use of the corresponding framework. Similarly, the parser associated with the language can easily be extracted from the tool source code. Any third-party developer should therefore be able to use it and modify it according to its requirements.

## 6.3 Limitations of our approaches

In Chapters 2 and in 3, we identified some limitations of the approaches we follow to compute the performance measures. We briefly recall these drawbacks in this section. Perspectives to overcome them are described in the next section.

The first limitation concerns the computation of the stationary probability vector of an inhomogeneous QBD. For recall, when the QBD has an infinite number of levels, the algorithm we presented requires to truncate the QBD at an arbitrary but large level. Selecting an adequate level is difficult, notably because it is case-specific and it can require a deep study of the considered system. Because of that, we have omitted to discuss the truncation procedure.

The second drawback concerns the simulation algorithm we developed. More precisely, we specified the terminating condition as a particular simulation time given by the user. Subsequently, when the simulation ends, we cannot be ensured that the steady state of the simulated process has been reached. Therefore, the influence of the warm-up period on the estimates may be too important. Two solutions are possible, namely

1. defining another terminating condition, or
2. using another simulation approach.

We briefly mention such an approach in the following section.

## 6.4 Perspectives and challenges

In this last section, we deal with the perspectives that follow our work and the challenges related to them. In particular, we mention limitations that we already discussed at the end of Chapter 4 and we give a lead about overcoming these ones.

As we stated in Section 4.6, specifying a QBD by using our textual language can be laborious. Indeed, it requires, for each type of transition to specify, to know explicitly the level and the phase of the QBD from which the transition starts, and the ones that are reached by the transition. The case study clearly illustrated this issue. Indeed, the state space of the studied system is composed of more than two dimensions. It is therefore difficult to determine the level and the phase of the QBD corresponding to a given state of the system. A modification of the language is therefore needed. For example, the user could be able to define rules to translate a state of an $n$-dimensional state space, with $n > 2$, to state space with two dimensions, namely the level and the phase.

We already mentioned that the matrix analytic methods are still intensively studied. In particular, algorithms related to a particular class of QBDs have been developed. For example, in the case of a sentivity analysis, Dendievel *et al.* [8] developed an expression to approximate the rate matrix associated with an homogeneous and infinite (see Section 2.1.3) with respect to a given perturbation. Because we chose to provide the matrix analytic methods as a framework, the integration of new algorithms should be eased for the developer and transparent to the user of the framework.

Like the matrix analytic methods, the simulation approaches still continue to be worked on by the scientific community. We mentioned in Section 4.6 the perfect simulation approach (see, *e.g.*, Thonnes [31]), which aims to remove the bias due to the warm-up period of the simulation. Relying on a perfect simulation in our simulation framework would allow us to overcome the major limitation of the framework, that is the uncertainty of reaching the steady state of the simulated process.

# Part V
# Appendix

# Appendix A

# ORBEL 25 Paper

As we mentioned in Chapter 4, we wrote two articles about our tool that were published in the proceeding of two conferences. ORBEL 25 is the first of these conferences. It is the 25th annual conference of the Belgian Operations Research Society. It was held at the university of Ghent, on February 10-11, 2011. The paper we include below is only an extended abstract. We gave a more thorough presentation at the conference, though.

# QBD Sensitivity Analysis Tool Using Discrete-Event Simulation and extension of `SMCSolver`

M. Cordy and M.-A. Remiche

University of Namur, Computer Science Faculty,
mcordy@student.fundp.ac.be, mre@info.fundp.ac.be

**Abstract**

We propose a tool that provides both analytical and simulation based performance analysis of both homogeneous and inhomogeneous Quasi-Birth-and-Death (QBD) processes. We extend SMCSolver in order to study inhomogeneous case and to analyze first passage times. Simulations are performed on a discrete-event based approach. We propose to analyze the sensitivity of a particular QBD.

**Keywords:** Markov process, Matrix Analytic Methods, SMCSolver

In the past, Quasi-Birth-and-Death (or QBD in short) processes have been extensively used for the design and the performance analysis of a great variety of systems, such as reliability systems, peer-to-peer systems, fluid Markov models or call center systems to cite but a few. QBD processes are particular cases of Markov processes, defined on bi-dimensional state space. The first dimension (often called the *level*) counts the number of occurrence of a stochastic event, while the second (called the *phase*) gives the state in which the whole system lies. Accordingly, the generator of this Markov process has the following form

$$Q = \begin{pmatrix} B_0 & B_1 & 0 & 0 & \dots \\ A_{-1}^{(1)} & A_0^{(1)} & A_1^{(1)} & 0 & \dots \\ 0 & A_{-1}^{(2)} & A_0^{(2)} & A_1^{(2)} & \dots \\ 0 & 0 & A_{-1}^{(3)} & A_0^{(3)} & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix},$$

where the inner blocks $A_i^{(k)}$ are of size $n_k \times n_{k+i}$, for $k$ positive integer and $i \in \{-1, 0, 1\}$, and $B_i$ is of size $n_0 \times n_i$, $i \in \{0, 1\}$. Diagonal elements of $B_0$ and of $A_0^{(k)}$, for all natural $k$ are negative. Other elements are positive or null. Moreover, the generator is such that the sum of its line gives 0.

When $A_i^{(k)} = A_i$ for all $k > 0$ and for all $i \in \{-1, 0, 1\}$ (expect for $A_{-1}^{(1)}$), the QBD is said to be *homogeneous*. Otherwise, the QBD is said *inhomogeneous*. In case the generator is of finite size; the QBD is said to be *finite*. Otherwise, the QBD is said *infinite*.

We refer to Latouche and Ramaswami [3] for a clear introduction to the matrix analytic methods that exist to perform QBDs analysis.

To perform sensitivity analysis of such systems is of great interest, in particular when measuring the robustness of optimal designed policies. Our aim is to propose a tool that would, as a first objective, rapidly give some insight about the sensitivity of QBD models subject to small variations on their input parameters. Our effort will be put on developing such a tool that will use either simulation or exact methods to solve the QBD, depending on the nature of the performance problem itself.

Exact computations are performed on both the original and perturbated QBD, except in the case of a $M/PH/1$ queue where we used results developed by Dendievel et al. in [2]. The tool also proposes to make use of a discrete-event based simulation procedure. This is of particular help when the size of the QBD is large. The tool supports the analysis of both homogenous and inhomogeneous QBDs. Input parameters can be specified following three different formats : the user may specify (i) the type of queue using Kendall's notation, (ii) the complete generator, (iii) the possible transitions.

There exists different tools to solve QBD, see for example MAMSolver developed by Riska and Smirni in [4], or SMCSolver proposed in [1]. We decide to incorporate the second tool to solve exactly homogenous QBD and to extend it to the analysis of inhomogeneous QBD.

## References

[1] D. Bini, B. Meini, S. Steffe, and B. Van Houdt. Structured markov chains solver : software tools. In *Proceedings of SMCTOOLS '06*. ACM Press, 2006.

[2] S. Dendievel, G. Latouche, and M.-A. Remiche. Perturbation analysis of an M/PH/1 queue. In *Performance 2010, posters*, 2010.

[3] G. Latouche and V. Ramaswami. *Introduction to Matrix Analytic Methods in Stochastic Modeling.* ASA-SIAM Series on Statistics and Applied Probability. SIAM, 1999.

[4] A. Riska and E. Smirni. Mamsolver: A matrix analytic methods tool. In *TOOLS '02: Proceedings of the 12th International Conference on Computer Performance Evaluation, Modelling Techniques and Tools*, pages 205–211, London, UK, 2002. Springer-Verlag.

# Appendix B

# ValueTools 2011 Paper

ValueTools is the name of the International ICTS Conference on Performance Evaluation Methodologies and Tools. Its 5th edition was held on May 16-20, 2011, at the ENS Cachan (France). We published another article that presents our tool in the conference proceeding and we also presented it during the conference. We include the article below.

# QBD Sensitivity Analysis Tool Using Discrete-Event Simulation and extension of *SMCSolver*

M. Cordy
The University of Namur
Faculty of Computer Science
Rue Grandgagnage, 21
5000 Namur, Belgium
maxime.cordy@fundp.ac.be

M.-A. Remiche
The University of Namur
Faculty of Computer Science
Rue Grandgagnage, 21
5000 Namur, Belgium
mre@info.fundp.ac.be

## ABSTRACT

We propose a tool that provides both analytical and simulation based performance analysis of both homogeneous and inhomogeneous Quasi-Birth-and-Death (QBD) processes. We extend SMCSolvers in order to study inhomogeneous case and to analyze first passage times. Simulations are performed on a discrete-event based approach. We also provide a rich input interface to give the most flexibility to the user to define its QBD transitions. The analysis of sensitivity in a complex level-dependent QBD model of a reliable system is an illustration of the wide range of QBD the tool may help to analyze.

## 1. INTRODUCTION

In the past, Quasi-Birth-and-Death (or QBD in short) processes have been extensively used for the design and the performance analysis of a great variety of systems, such as reliability systems (see [13] for example), peer-to-peer systems (see [5]), fluid Markov models (see [3] among many others) or call center systems (see [7]) to cite but a few. QBD processes are particular cases of Markov processes, defined on state space with the following structure

$$\mathcal{S} = \{(k,i); k \in \mathbb{N}, 0 \leq i \leq n_k\}, \tag{1}$$

with usually $n_k < \infty$ for all $k \in \mathbb{N}$. Accordingly, their generator has the following form

$$Q = \begin{pmatrix} B_0 & B_1 & 0 & 0 & \dots \\ A_{-1}^{(1)} & A_0^{(1)} & A_1^{(1)} & 0 & \dots \\ 0 & A_{-1}^{(2)} & A_0^{(2)} & A_1^{(2)} & \dots \\ 0 & 0 & A_{-1}^{(3)} & A_0^{(3)} & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}, \tag{2}$$

where the inner blocks $A_i^{(k)}$ are of size $n_k \times n_{k+i}$, for $k \in \mathbb{N}_0$ and $i \in \{-1,0,1\}$, and $B_i$ is of size $n_0 \times n_i$, $i \in \{0,1\}$. Diagonal elements of $B_0$ and of $A_0^{(k)}$, for all $k \in \mathbb{N}$ are negative. Other elements are positive or null. Moreover, we have

$$B_0\vec{1} + B_1\vec{1} = \vec{0} \tag{3}$$

$$A_{-1}^{(k)}\vec{1} + A_0^{(k)}\vec{1} + A_1^{(k)}\vec{1} = \vec{0} \tag{4}$$

for all $k \in \mathbb{N}$, where $\vec{1}$ and $\vec{0}$ respectively are vectors full of 1 and 0 respectively and of appropriate size.

When $A_i^{(k)} = A_i$ for all $k \in \mathbb{N}_0$ and for all $i \in \{-1,0,1\}$ (expect for $A_{-1}^{(1)}$), the QBD is said to be *homogeneous*. Otherwise, the QBD is said *inhomogeneous*. In case there exists $K < \infty$, such that

$$Q = \begin{pmatrix} B_0 & B_1 & 0 & \dots & & \\ A_{-1}^{(1)} & A_0^{(1)} & A_1^{(1)} & \dots & 0 & 0 \\ 0 & A_{-1}^{(2)} & A_0^{(2)} & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & B_{-1} & B_K \end{pmatrix}, \tag{5}$$

with $B_{-1}$ of size $n_K \times n_{K-1}$ and $B_K$ of size $n_K \times n_K$; the QBD is said to be *finite*. In case it does not exists such a $K$, the QBD is said *infinite*.

We refer to Latouche and Ramaswami [8] for a clear introduction to the matrix analytic methods that exist to perform QBDs analysis.

To perform sensitivity analysis of such systems is of great interest, in particular when measuring the robustness of optimal designed policies. Our aim is to propose a tool that would, as a first objective, rapidly give some insight about the sensitivity of QBD models subject to small variations on their input parameters. Our effort will be put on developing such a tool that will use either simulation or exact methods to solve the QBD, depending on the nature of the performance problem itself.

At this stage, exact computations are performed on both the original and perturbated QBD, except in the case of a $M/PH/1$ queue where we used results developed by Dendievel et al. in [4]. The tool also proposes to make use of a discrete-event based simulation procedure. This is of particular help when the size of the QBD is large. The tool supports the analysis of both homogenous and inhomogeneous QBDs. Input parameters can be specified following three different formats : the user may specify (i) the type of queue using Kendall's notation, (ii) the complete generator,

(iii) the possible transitions.

There exists different tools to solve QBD, see for example MAMSolver developed by Riska and Smirni in [14], or SMC-Solver proposed in [2]. We decide to incorporate the second tool to solve exactly homogenous QBD and to extend it to the analysis of inhomogeneous QBD. We also propose to add a function to compute first passage times.

Such tools usually propose to specify the blocks that compose the QBD generator (see Equation (2)). As mentioned earlier, our tool allows the user to specify the input parameters by means of transition specification. This method is indeed necessary to implement in order to offer the possibility to deal with general structured inhomogeneous QBDs. In Katoen et al. [6], all possible transitions have to be given by the user. In our tool, repetitive structure of the inner blocks can be defined as such.

The paper is composed of three main sections. First, we explain the tool design choices operated for both the input and output interfaces and the architecture of the code itself. In the following section, we perform as an illustration to our work, a sensitivity analysis of a preventive repair policy in a reliable system. This model was first discussed in [12]. Finally, we conclude our work by indicating work that need to be done in the future.

## 2. TOOL ARCHITECTURE

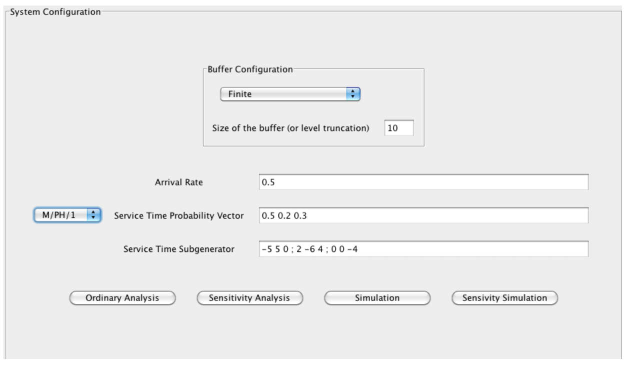
In this section, we discuss the specificity of the tool regarding the input and output interfaces, as well as the architecture of the code itself.

### 2.1 Input interface

An interesting feature of the tool is the ability for the user to specify the QBD generator by using one of the three different input interfaces. We now describe each method.

The first possibility is to define the QBD as a queueing system, using Kendall's notation in concise form. After choosing the queue type, each parameter of the selected system, such as an exponential rate or a phase-type distribution, must be entered. We refer to Latouche and Ramaswami [8], Chapter 2 for a clear introduction to phase-type distributions. At the moment, the tool is restricted to the $M/M/1$, $M/PH/1$, $PH/M/1$ and $PH/PH/1$ queues. However, it can be extended to be able to work with more queue types. The size of the buffer, possibly infinite, can also be chosen. Figure 1 illustrates how our input interface looks like. In this example, we want to specify an M/PH/1 queue. We must define the size of the buffer, the arrival rate, the probability row vector and the generator matrix of the phase-type distribution. In this particular example, the generator is a $3 \times 3$ matrix, where each component of a row is separated by a comma and each line of the matrix is separated by a semicolon.

The second input interface allows to explicitly define the blocks that compose the generator. For this purpose, the user must provide a Matlab or Octave function. **Octave** is a programming language specialized in numerical computations. Its syntax and its semantic are almost identical to the



System Configuration

Buffer Configuration

Finite

Size of the buffer (or level truncation): 10

M/PH/1

Arrival Rate: 0.5

Service Time Probability Vector: 0.5 0.2 0.3

Service Time Subgenerator: -5 5 0 ; 2 -6 4 ; 0 0 -4

**Figure 1: Definition of a QBD generator as a queueing system, using Kendall's notation.**

MATLAB ones. Octave is a free software under the terms of the GNU General Public License.

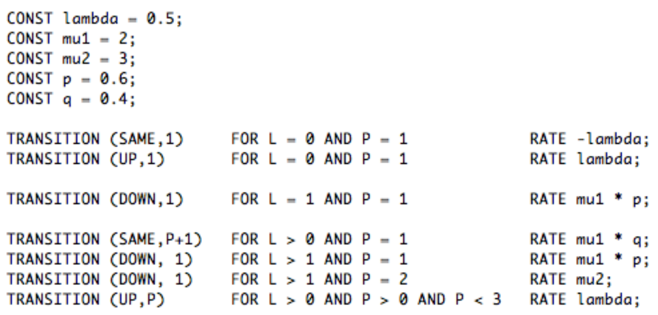
This function would have one parameter: that is $i$, the level for which the inner blocks would be computed. The output parameters of the function are then the three inner blocks corresponding to this level, given in the following order: $A_{-1}^{(i)}$, $A_0^{(i)}$ and $A_1^{(i)}$. If one of these three matrices is not defined for the considered level, the function returns an empty matrix. For example, the result of the function for level 0 would be: [], $B_0$, $B_1$ as specified in Equation (2).

For the program to load the function, the user has to provide the path where to find it. Once the function loaded, the tool has everything it needs to define the QBD process. The main drawbacks of this method are that first programming in Octave is required. Second, knowing the exact and complete form of the generator is required, while it may be easier to define only the transitions of the corresponding process. The next and last method is an answer to these problems.

The third and last input method consists in specifying the possible transitions of the process in a syntax consistent and textual language we developed. The particular feature of our method is that all the possible transitions have not to be given. Instead, the user may specify more concisely a repetitive structure appearing in the generator. Our tool is now restricted to two-dimensional Markovian state spaces. A small context-free grammar has thus been developed in order to build our parser. It reads a text file respecting this grammar and accordingly, produces an Octave function. As for the second method, this function returns the inner blocks of a given level. For readability reason, we present in the appendix the most important symbols of the grammar in Backus-Naur Form, as well as their semantic.

Figure 2 illustrates this specification method for the definition of an M/PH/1 queue. The arrival rate is *lambda.* The service time distribution is completely specified using four parameters, these are *mu1, mu2, p* and *q*. As specified in our code, these input parameters are *constant* in the specification of the transitions.

We are then able to specify the *transitions.* Each transition definition begins with the keyword `TRANSITION`, followed by



```
CONST lambda = 0.5;
CONST mu1 = 2;
CONST mu2 = 3;
CONST p = 0.6;
CONST q = 0.4;

TRANSITION (SAME,1)      FOR L = 0 AND P = 1             RATE -lambda;
TRANSITION (UP,1)        FOR L = 0 AND P = 1             RATE lambda;

TRANSITION (DOWN,1)      FOR L = 1 AND P = 1             RATE mu1 * p;

TRANSITION (SAME,P+1)    FOR L > 0 AND P = 1             RATE mu1 * q;
TRANSITION (DOWN, 1)     FOR L > 1 AND P = 1             RATE mu1 * p;
TRANSITION (DOWN, 1)     FOR L > 1 AND P = 2             RATE mu2;
TRANSITION (UP,P)        FOR L > 0 AND P > 0 AND P < 3   RATE lambda;
```

**Figure 2: Definition of an M/PH/1 queue using text-based transition specification.**

the type of transition and the destination phase. More precisely,

- possible type of transitions are `UP`, `DOWN` or `SAME` respectively, which describes movement to the upper level, the downer level or in the same level, respectively and
- the phase is explicit or is given through some constraint.

We then give these constraints for the level `L` and the phase `P`. The instruction finishes when the *rate* is finally given using possible predefined constants.

In Figure 2, we observe that the first specified transition means that when in level 0 and phase 1, the process may stay is that state for an exponential amount of time whose rate is `lambda`. We also observe that the fourth specified transition concerns all the level strictly greater than 0, but only phase 1. It indicates that in that starting state, the process may move to state `(L,P+1)` with a rate of probability `mu1` $\times$ `q`.

Once the QBD is defined, the tool may analyze it using matrix analytic methods, or may simulate it via discrete-event simulation and accordingly, carry out a sensitivity analysis. The user indicates his choice by clicking on the corresponding button. When simulating it, some additional parameters must be set, such as the starting level, the starting phase and the simulation end time.

When doing sensitivity analysis, the tool will work with a perturbed generator defined as follow:

$$Q_{pert} = Q + \epsilon\tilde{Q} \tag{6}$$

Firstly, the perturbation generator $\tilde{Q}$ must be specified via the same input interface that was used to define the QBD generator $Q$. After that, a set of values that $\epsilon$ will take must be chosen. Then, for each selected value of $\epsilon$, the tool will perform an analysis or a simulation on the resulting $Q_{pert}$, whose inner blocks will be computed dynamically.

## 2.2 Output interface and performed analysis

The tool computes different data that may be useful when evaluating the performance of a QBD process. In this first version of the tool, we focus on the stationary probability vector $\vec{\pi}$ and first passage times.

Following the matrix analytic methods, key matrices $R_i$ (with $i$ being the level) must be computed in order to determine the stationary probability vector. In the particular case of an homogeneous QBD, we have used the SMCSolver ([2]) implementation of the *Logarithmic Reduction* algorithm (see [8], Chapter 8), which is quadratically convergent. To handle the level-dependent case, we have implemented in Octave the algorithm presented in [8], Chapter 12.

In [8], Chapter 11, two algorithms to compute the expected first passage time from the level 0 to any upper level of an homogeneous QBD are detailed: these are the *Linear Level Reduction* and the *Reduction by Bisection* algorithms. We have chosen to implement the former, as it can be more easily generalized to compute the expected first passage time from any level to any other one, in both the homogeneous and inhomogeneous cases. Our tool implements that generalization.

In case of a simulation, we use a discrete-event simulation approach (see Leemis and Parkin [9]). The main lines of the algorithm are as follows, assuming the process is in level $i$, phase $\phi$,

- the inner blocks $A_{-1}^{(i)}, A_0^{(i)}, A_1^{(i)}$ corresponding to the current level $i$ are obtained.
- These blocks are discretized with a rate $r$ equal to the maximum absolute value of the diagonal of $A_0^{(i)}$. We follow here the principles of the uniformization method (as described in Latouche and Ramaswami [8], Section 2.8).
- Next only the line corresponding to the current phase ($\phi$) is considered. It now gives the transition probabilities that after a random exponential time $t$ (with rate $r$), the process moves to another state or remains in state $(i, \phi)$. We simulate that transition.
- Accordingly, the process moves to this state and the simulation time is increased by $t$.

We repeat this procedure until the simulation end time is reached.

Note that the algorithm covers both the homogeneous and inhomogeneous cases. However, for the homogeneous case, an optimization can be made by computing only one time the inner blocks as those are identical for each level, except for the levels 0, 1 and the two last levels in case of a finite QBD.

The stationary probability of a given state is then estimated by dividing the time spent in this state during the simulation by the total simulation time. We also provide the confidence interval round our estimation (see Figure 3 for one particular M/PH/1 case).

First passage times are obtained as arithmetic mean of time at which the simulation reached the level of interest. Enough

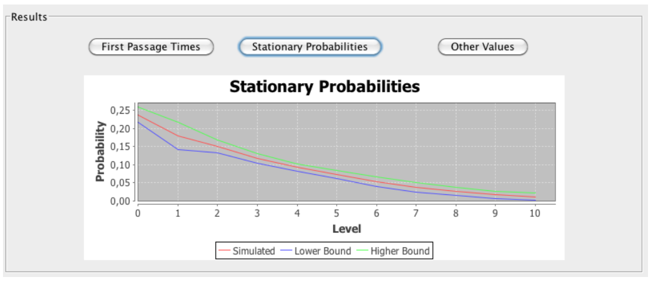


**Figure 3: Simulation results for the stationary probability of a M/PH/1 case.**

simulations must be performed and confidence intervals are always provided to measure it.

The outputs are given either in graphical format or as a text file. The text file presents at each line the steady state probabilities of the corresponding level. When a plot is provided, $X$axis gives the level while $Y$axis gives the probability that the process is in level $i$, whatever the phase (see Figure 3 for an example of the graphical output interface). Finally, the expected first passage time starting in a given level $i$ can also be plotted in a graph whose abscissa is the target level $j$ and the ordinate is the expected first passage time from $i$ to $j$. The expected first passage times from $i$ can also be given via in a text file. In this case, for each destination level $j$, we provide the expected first passage time to that level for each possible phase of the starting level $i$.

As mentioned before, the tool allows to do some sensitivity analysis either by using matrix analytic methods or simulations on the perturbed process. The same data can then be computed for each $\epsilon$ value of the perturbation. Then, a graph showing the influence of the perturbation on a specific performance measure (that is the stationary probability of a particular level or the first passage time from a given level to another one) can be displayed. The $X$axis gives the different values $\epsilon$ takes. The $Y$axis is the value of the considered performance measure.

## 2.3 Code architecture

The tool is composed of some Java and GNU Octave modules. One of our goals is to make it quite easily modifiable. Therefore, we focus on modularity by dividing the program into high-level components. Figure 4 shows those different components and the dependencies between each other. Each green box represents a component. The dependencies are indicated by the arrows. The component at the origin of an arrow depends on the component at the edge of this arrow. Before describing each component, its roles and dependencies, we first motivate the choice of the programming language.

Each **.m** file contains all the Octave functions implementing the matrix-analytic methods and simulation algorithms. Some of them come from SMCSolver [2]. We choose to provide these functions as a stand-alone resource so that one may be able to call them directly with Octave, instead of only observing their results via the graphic user interface developed in Java.

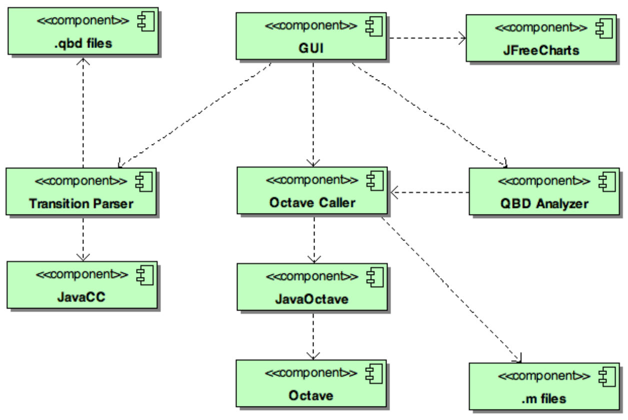


**Figure 4: High-level architecture of the tool.**

We choose to use **Octave** as the computation core of our tool because of its higher performance in numerical (mainly matrix) computation, compared to Java. Two features of this language must be kept in mind:

- Octave uses an interpreter to execute instructions which are written through its command-line interface or are contained in a script.
- It allows to dynamically load some new functions or to redefine an existing one.

**JavaOctave** is a Java library that works as a bridge from Java to Octave. It was developed by Kim Hansen[1]. It allows to call the Octave interpreter from a Java program and to transform an Octave data structure into a Java object. Basically, it runs Octave and provides some functions to send back the instructions written in the Octave language.

Let us now present some of the main components in Figure 4.

**OctaveCaller** is a Java component we developed. It is the only component that communicates with JavaOctave. Thanks to this property, modularity is improved: the other Java components are completely independent from Octave components. Since JavaOctave is a bridge between Java and Octave, OctaveCaller can be seen as a gate on the Java side. It provides routines to keep track of every Octave function that has been declared. Any function can thus be called and its return values will be contained in Java objects managed by the OctaveCaller.

The Java component **QBD Analyser** is the masterpiece of the tool. It determines which function must be called, depending on the chosen analysis method, the characteristic of the defined QBD process and the desired output data. It determines which function has to be registered in the OctaveCaller, when and how such a function must be modified. It also sets the right parameters of a function and orders the OctaveCaller to call it. Then, it analyzes the data structures returned by the OctaveCaller to extract the desired results.

[1]http://kenai.com/projects/javaoctave/

The general performance of the tool highly depends on the choices of the QBD Analyzer. For example, when a simulation of an homogeneous QBD is required, it can either choose to call the general function that can simulate any QBD or the specific function for the level-independent case. The latter is far more efficient than the former, as the inner blocks of the QBD are computed once at the start of the simulation instead of being determined at each level movement.

**JavaCC**[2] is an open source tool originally owned by Sun Microsystems. Basically, it allows to generate a lexical analyzer and a descending syntax analyzer from a conflict-free grammar defined in Backus-Naur Form (see [1] Chapters 1 to 4, for a clear introduction). A FAQ[3] about JavaCC is also maintained by Theo Norvell at Memorial University of Newfoundland.

We used JavaCC to create the **Transition Parser** component. Its role is to parse a text file (a **.qbd** file) respecting the grammar we developed (see Section 2.1 and Appendix A) and to produce an Octave function that dynamically computes the inner blocks of the QBD corresponding to the defined transitions. Accordingly, the input grammar and the output language are completely independent.

**JFreeChart**[4] is a free library that allows the developers to display graphs inside their Java applications. It is distributed under the GNU Lesser General Public License. It supports a large variety of graph types and provides a complete API for dynamically editing the graph, as well as performing zooms on it.

Finally, the **Graphic User Interface (GUI)** component has two main roles. Firstly, it displays the different windows through which the user can navigate. That includes both the input windows, in which the parameters of the studied system are entered and the path to essential files, as well as the output windows in which the evaluation results are displayed. Secondly, it manages the user actions and requests the right service of the right component when necessary. Basically, this component reads the data introduced by the user, calls the right input interface in order to set every parameter and ask the QBD Analyser to perform the analysis.

## 3. SENSITIVITY ANALYSIS OF A RELIABLE SYSTEM

As an illustration to our tool, we propose to analyze the sensitivity of a particular reliable system, as first defined and analyzed in [12]. We choose this model as it constitutes one clear example of the use of inhomogeneous finite QBD (see Equation (5)) to model a complex system. It thus allows us to clearly highlight the use and the versatility of our tool.

In this section, we explain the system itself. We then clarify the state-space's definition as exposed in [12]. We propose in this summary, to give the inner structure of the $A_0^{(i)}$, $1 \leq i \leq n-1$ only. We choose the second method to specify the structure of the generator to the tool. Finally, we propose to measure the sensitivity of the system subject to longer or shorter inspection.

[2]https://javacc.dev.java.net/
[3]http://www.engr.mun.ca/~theo/JavaCC-FAQ/
[4]http://www.jfree.org/jfreechart/

### System definition

The system is composed of $n$ units. One unit is online, the others are in warm standby. However units are subject to degradation and eventually may go to *corrective* repair. Only one unit may be repaired at a time, others are queueing in FIFO order. To prevent full degradation when online, inspections are randomly performed. In case the level of degradation is too high, a *preventive* repair is performed, except if the system contains no more standby unit. Would the system be empty of standby units and the online unit need a corrective repair, one unit in preventive repair would be preempted (if available). In the other case, the system would be said to have failed.

Standby and online lifetimes (prior to full degradation) respectively are assumed to be phase-type distributed PH($\alpha_s$, $T_s$) of order $m_s$ and PH($\alpha, T$) of order $m$ respectively. Preventive and corrective repairs take a random phase-type distributed time, with parameter $(\beta_p, S_p)$ of order $n_p$ and $(\beta_c, S_c)$ of order $n$ respectively. Inspection times are random and two consecutive inspection procedures are separated by a phase-type random time with parameters $(\gamma, L)$ of order $\nu$. All random variables are independent of each other.

### State-space definition and decomposition

As estabished in [12], the system can be modeled as a Markov process whose state-space is

$$(i, j, k, z^{(i,j)}, l_p, l_c, f) \tag{7}$$

where

- $i$ is the number of units in preventive repair, with $0 \leq i \leq n-1$,
- $j$ is the number of units in corrective repair, with $0 \leq j \leq n$,
- $k$ is the phase occupied by the online unit, with $1 \leq k \leq m$,
- $z^{(i,j)}$ is the phase of the standby units,
- $l_p$ is the phase of the unit in preventive repair, with $1 \leq l_p \leq n_p$,
- $l_c$ is the phase of the unit in corrective repair, with $1 \leq l_c \leq n_c$,
- $f$ is the phase of the inspection procedure, with $1 \leq f \leq \nu$.

Let us be more precise about $z^{(i,j)}$. This vector is composed of $(z_1, z_2, \ldots, z_{n-1-(i+j)})$, with $1 \leq z_r \leq m_s$, where $1 \leq r \leq n-1-(i+j)$.

The *level* $M$ is defined as the number of units in repair (either preventive or corrective repair), that is $0 \leq M \leq n$, with

$$M = \{(i,j); 0 \leq i \leq n-1 \text{ and } j = M - i\}, \tag{8}$$

$$M = \{(0,n)\}. \tag{9}$$

We now explain the inner structure of one particular block, that is $A_0^{(M)}$, for $2 \leq M \leq n-2$. Other blocks that compose generator (5) are clearly defined in Appendix A of [12]. Our objective is here to illustrate the complexity of the inhomogeneous QBD that may be handled by our tool.

We have

$$A_0^{(M)} = \begin{pmatrix} A_0^{(M)}(1,1) & & \\ & A_0^{(M)}(2,2) & \\ & & A_0^{(M)}(3,3) \end{pmatrix}, \qquad (10)$$

where level $M$ has been partitioned in three subsets, that is $M = M_1 \cup M_2 \cup M_3$, defined as follows

$$M_1 = \{(0, M)\} \qquad (11)$$
$$M_2 = \{(i,j); 1 \leq i \leq M-1, j = M-i\} \qquad (12)$$
$$M_3 = \{(M, 0)\}. \qquad (13)$$

This explains why only diagonal blocks are non-null. Indeed, moving from a state in $M_i$ to a state in $M_j$ $(j \neq i)$ implies some repair to be finished and a new one to start.

We then have for matrix $A_0^{(M)}(1,1)$

$$\begin{aligned} A_0^{(M)}(1,1) = & \; T \otimes I_{(m_s)^{n-M-1} n_c \nu} \\ & + I_m \otimes (T_s \oplus \ldots \oplus T_s) \otimes I_{n_c \nu} \\ & + I_{m (m_s)^{n-M-1}} \otimes S_c \otimes I_\nu \\ & + I_{m(m_s)^{n-M-1} n_c} \otimes L \\ & + U_1 \otimes I_{(m_s)^{n-M-1} n_c} \otimes L^0 \gamma, \end{aligned} \qquad (14)$$

where $I_n$ is the identity matrix of size $n$,

$$U_1 = \begin{pmatrix} I_g & 0 \\ 0 & 0 \end{pmatrix}_{m \times m} \qquad (15)$$

is a matrix that permits to identify the states in which the units do not need to go to corrective repair (the first $g$ phases are ok, others are not), and

$$L^0 = -L\,\vec{1}. \qquad (16)$$

Equation (14) is readily explained as follow. No change of level in this Markov process implies that we did observe a change of phase only. This can be a change of phase for the online unit (determined by $T$) or ("+") for one of the standby units (determined by $T_s$) or ("+") for the corrective repair unit (determined by $S_c$) or ("+") for the inspection unit (determined by $L$). In case a new inspection period starts (determined by $L^0\gamma$), the online unit was not in a too degraded state (determined by $U_1$).

In our tool, we choose the second input interface to specify these matrices. This one example showed the complexity of this inhomogeneous QBD. The best is to code all the matrices according to an Octave program and to let our program load it and call it when necessary.

## Sensitivity analysis

We propose to identify the impact of a shorter or longer inspection period on the rocof of the system, that is the probability that the system fails completely (i.e. no more unit is available to become the online unit).

Authors in [12] had proposed such a study based on the phase $g$ (see $U_1$ definition in (15)) at which the unit needed to go to preventive repair. They were able to measure the prize of being more strict on the need to go preventive repair. We wish here to see if performing more often inspection could have the same effect on the rocof of the system. For this we choose a perturbation of the inspection procedure as follows

$$L_\epsilon = L + \epsilon A \qquad (17)$$

where

$$A = \begin{pmatrix} -1 & 1 \\ 0 & -1 \end{pmatrix}, \qquad (18)$$

and $\epsilon$ ranges from 0.01 to 0.09 by step of 0.02.

We choose exactly the same input parameters as they did in [12] and obtain the results in Table 1.

We have chosen $\epsilon$ such that the greater $\epsilon$, the smaller the interval in between two consecutive inspections. Accordingly, we observe in Table 1 that for a given $g$, the greater $\epsilon$, the smaller the rocof. Indeed, the system will repair more rapidly the default units. As established in [12] the greater $g$ the greater the rocof. This makes sense since on the contrary in this case, the inspection will cause a preventive repair more lately. With this sensitivity analysis, we may now decide about a compromise in between the phase of decision for preventive repair (that is $g$) and the rate of inspection, that is

$$\mu = \gamma(-(L + \epsilon A))^{-1}\vec{1}. \qquad (19)$$

## 4. PERSPECTIVE AND FUTURE WORK

There are two main directions we wish to follow in the future to extend our tool. First, we wish to integrate recent and further developments on sensitivity analysis of QBD process. Second, the grammar need to allow the user to define more complex QBD transition structure and state space.

Sensitivity analysis is the subject of many research papers (see for example [10] or [11]) but few propose a systematic and tractable approach to any kind of QBD. In our tool, at this stage of the development, we have chosen to carry out two type of analysis, one on the original QBD and one on the perturbated QBD. This is clearly not efficient and our tool will definitely need to integrate recent advances in this matter.

Future work on the grammar should permit to extend our approach to $n$-dimensional Markovian state-space. We should also be able to cover the case where complex dependencies in between state transition and rate of transition occur.

Let us conclude that recent developments on QBD simulations techniques (such as perfect simulation for example) might be of great interest to be included in our tool.

## APPENDIX

## A. DEFINITION OF THE GRAMMAR USED FOR THE INPUT INTERFACE

We develop a small context-free grammar in order to build a parser that reads a text file respecting this grammar and

**Table 1: Performance analysis on the rocof of the system**

| $g$/ $\epsilon$ | 0.01 | 0.03 | 0.05 | 0.07 | 0.09 |
|---|---|---|---|---|---|
| 1 | 1.4921e-06 | 8.3196e-07 | 6.3283e-07 | 5.4581e-07 | 4.9886e-07 |
| 2 | 2.4186e-06 | 1.5270e-06 | 1.2121e-06 | 1.0623e-06 | 9.7770e-07 |
| 3 | 5.9177e-06 | 4.4985e-06 | 3.8340e-06 | 3.4591e-06 | 3.2224e-06 |
| 4 | 1.5043e-05 | 1.4081e-05 | 1.3302e-05 | 1.2666e-05 | 1.2141e-05 |

constructs the generator of a QBD process. This appendix presents all the symbols composing the grammar as well as their informal semantic. It also aims to explain how to specify the transitions of a QBD thanks to them. Let us recall that our approach is limited to two-dimensional Markovian state spaces. All the grammar symbols are defined in figure 5.

Our QBD specification method consists in the declaration of a number of constants followed by the declaration of a number of transitions. We propose to consider Figure 2 as a clear example of the use of the grammar. While it is allowed not to declare any constant, we have imposed the restriction that at least one transition must be defined.

The declaration of a *constant* begins with the keyword `CONST`, which is followed by an *id* and then a *value*. Variable *id* simply represents the name of the constant. It is a string beginning with a lower case letter. This letter can be followed by a series of some of the following characters: letters, digits, underscore. A *value* is a decimal number and represents the value of the constant.

The definition of a transition begins with the keyword `TRANSITION`, followed by the type of transition and the destination phase. More precisely, the possible types of transition are `UP`, `DOWN` and `SAME` respectively, that describes if the process moves to the upper level, the downer level or stays in the same level, respectively. The destination phase has either explicit value or depends on the starting phase $P$.

Then, a series of constraints on the starting level $L$ and starting phase $P$ are specified. They are preceded by the keyword `FOR` and separated from each other by the keyword `AND`. The conjunction of these constraints explains for which source states the transition is defined. Finally, the keyword `RATE` is followed by an expression giving the rate at which the transition occurs. This can be a simple expression such as a decimal number, an *id* referencing a constant. It can also be compound, i.e. simple expressions joined by an arithmetic operator ($+, -, *$ and $/$) or by the binary function `min` and `max`. Finally, expressions can be grouped using parentheses.

Accordingly, one could define the next transition using our grammar by writing:

```
TRANSITION (UP,1) FOR L > 1 AND P = 3 RATE 5.2
```

It means that the process can move from every state $(L, P)$ such that $L > 1$ and $P = 3$ to the upper level and in phase 1, thus to the state $(L + 1, 1)$ with rate 5.2.

## B. THE OCTAVE FUNCTIONS

```
qbd          ::=  (constant)* (transition)+
constant     ::=  CONST id value
transition   ::=  TRANSITION (move, expression)
                  FOR conditions RATE expression
move         ::=  UP | SAME | DOWN
conditions   ::=  condition (AND condition)*
condition    ::=  (L | P) (< | <= | = | >= | >) expression
expression   ::=  term term_0
term_0       ::=  ((+ | -) expression) | ϵ
term         ::=  factor factor_0
factor_0     ::=  ((* | /) term) | ϵ
factor       ::=  id | L | P | value
                  | ( expression )
                  | - expression
                  | min( expression , expression )
                  | max( expression , expression )
id           ::=  [a-z]([a-z] | [A-Z] | [0-9] | _])*
value        ::=  (0 | [1-9] [0-9]*) ([.][1-9][0-9]*)?
```

**Figure 5: Definition of the symbols of the context-free grammar. $\epsilon$ represents the empty symbol.**

We use Octave as the computation core of our tool. Therefore, we provide some Octave functions that implement the simulation and the matrix-analytic algorithms. These functions can be either called from the Octave command line or from our Java code. In this appendix, each function is described. We explain specifically what are their parameters and their return values.

The functions can be separated into two groups. The first group is a set of functions implementing the matrix-analytic algorithms to compute the rate matrices, the stationary probability vector or the expected first passage times. The second group includes the simulation functions, as well as other functions used to provide some statistics from the simulation results.

### *Matrix-analytic functions*

One of our goal is to compute the stationary probability vector. In the infinite and homogeneous case, we use some functions of the SMCSolver (see [2]): **QBD_LR.m** and **QBD _pi.m**. The first one computes the rate matrix $R$ by using the *Logarithmic Reduction* algorithm. The second one computes the steady state vector. In case of finite and inhomogeneous QBDs respectively, we define two other functions: **QBD_pi_finite.m** and **QBD_pi_Inh.m** respectively. Both implement a modified version of the *Linear Level Reduction* algorithm ([8], Chapters 10 and 12). One key difference between these two functions is how they get the inner blocks. In the former function, they are passed as parameters. Thus, the arguments of **QBD_pi_finite.m** are: $A_{-1}^{(1)}$, $B_0$, $B_1$, $A_{-1}$, $A_0$, $A_1$, $A_{-1}^{(K)}$, $A_0^{(K)}$, $A_1^{(K-1)}$ and $K$ (as defined in Equation (5)). In the latter case, that is **QBD_pi_Inh.m**, the inner blocks are dynamically computed by calling another function that returns those and that takes only one parameter:

the level to compute the block of. This function must be implemented in the file **computeLevelMatrices.m**.

Once the steady state vector is computed, it can be passed as a parameter of **QBD_stat.m**. This function returns the mean stationary visited level and its standard deviation.

To compute the expected first passage times from a given level to an upper one, an algorithm was proposed in [8], Chapter 11. The function **QBD_fpt_LLR.m** implements a generalized version of this algorithm. It allows to compute the expected first passage times from every phase of a given level to any another level. It takes two parameters: $s$, the origin level and $d$, the destination level. The inner blocks are dynamically computed whenever they are needed thanks to **computeLevelMatrices.m**.

### *Simulation functions*

The discrete-event simulation algorithm is implemented via three functions. Again, these functions only differ in the way the inner blocks are obtained.

**QBD_sim_hom.m** simulates an infinite homogeneous QBD. Six inner blocks are needed: $A_{-1}^{(1)}$, $B_0$, $B_1$, $A_{-1}$, $A_0$ and $A_1$. They are computed and stored before the simulation begins.

To simulate an homogeneous and finite QBD, three more inner blocks are needed: $B_{-1}$, $B_K$, $A_1^{(K-1)}$ (if different from $A_1$). Along with the six previously defined blocks, they can be passed as parameters of the function **QBD_sim_hom_fin.m**.

Finally, the inhomogeneous case is simulated by the function **QBD_sim_Inh.m**. The inner blocks of a given level are computed when this level is reached for the first time. Then, these blocks are stored in order to be used whenever they are needed. Thus, a block is computed at most one time.

Each simulation function computes the estimated stationary probability of every state of the process, the mean visited level, its standard deviation, the lowest and the highest reached levels.

The first passage time from the starting level to any reached level is also returned as a vector.

If the number of batches is greater than one, a confidence interval for the expected probability of every state is also computed. The lower (respectively higher) bounds are contained in the returned value *lowerBounds* (respectively *higherBounds*).

This interval is obtained by calling **Estimate_Mean.m**. This function has two parameters: *Sample*, a vector that contains the sample values, and *alpha*, the level of confidence of the estimation.

## C. REFERENCES

[1] A. Aho, R. Sethi, and J. Ullman. *Compilers: Principles, Techniques, and Tools.* Addison-Wesley, 1985.

[2] D. Bini, B. Meini, S. Steffe, and B. Van Houdt. Structured markov chains solver : software tools. In *Proceedings of SMCTOOLS '06*. ACM Press, 2006.

[3] A. da Silva Soares and G. Latouche. Matrix-analytic methods for fluid queues with finite buffers. *Performance Evaluation*, 63:295–314, 2006.

[4] S. Dendievel, G. Latouche, and M.-A. Remiche. Perturbation analysis of an M/PH/1 queue. In *Performance 2010, posters*, 2010.

[5] S. Hautphenne, K. Leibnitz, and M.-A. Remiche. Modeling of P2P file sharing with a level-dependent QBD process. In W. Yue, Y. Takahashi, and H. Takagi, editors, *Advances in Queueing Theory and Network Applications*, pages 247–263. Springer, 2009.

[6] J.-P. Katoen, M. Khattri, and I. S. Zapreev. A Markov reward model checker. In *Quantitative Evaluation of Systems (QEST)*, pages 243–244, Los Alamos, CA, USA, 2005. IEEE Computer Society.

[7] K. Kawanishi. QBD approximations of a call center queueing model with general impatience distribution. *Computers & Operations Research*, 35:2463–2481, 2008.

[8] G. Latouche and V. Ramaswami. *Introduction to Matrix Analytic Methods in Stochastic Modeling.* ASA-SIAM Series on Statistics and Applied Probability. SIAM, 1999.

[9] L. Leemis and S. Park. *Discrete-event Simulation. A First Course.* Pearson, 2006.

[10] Q.-L. Li and L. Liu. An algorithmic approach for sensitivity analysis of perturbed quasi-birth-and-death processes. *Queueing Systems*, 48:365–397, 2004.

[11] C. D. Meyer. Sensitivity of the stationary distribution of a Markov chain. *SIAM J. Matrix Anal. Appl.*, 15:715–728, 1994.

[12] D. Montoro Cazorla and R. Pérez-Ocón. An ldqbd process under degradation, inspection, and two types of repair. *European Journal of Operational Research*, 190(2):494–508, 2008.

[13] R. Pérez-Ocón and D. Montoro-Cazorla. A multiple system governed by a quasi-birth-and-death process. *Reliability Engineering and System Safety*, 84:187–196, 2004.

[14] A. Riska and E. Smirni. Mamsolver: A matrix analytic methods tool. In *TOOLS '02: Proceedings of the 12th International Conference on Computer Performance Evaluation, Modelling Techniques and Tools*, pages 205–211, London, UK, 2002. Springer-Verlag.

# Appendix C

# Case study: original article

The case study we carry out in Chapter 5 is based on an already existing case study, originally made by Montoro and Perez [23]. Because the building of the infinitesimal generator of the Quasi-Birth-and-Death process modelling the studied system is too laborious to be presented thoroughly, we only discussed the building of some of the inner blocks. In order to give the complete generator, we include in this appendix the original article of Montoro and Perez [23].

Available online at www.sciencedirect.com

ScienceDirect

European Journal of Operational Research 190 (2008) 494–508

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/ejor

Stochastics and Statistics

# An LDQBD process under degradation, inspection, and two types of repair

Delia Montoro Cazorla [a], Rafael Pérez-Ocón [b,*]

[a] *Dep. de Estadística e I.O., Universidad Jaén, España, Spain*
[b] *Dep. de Estadística e I.O., Universidad de Granada, España, Spain*

Received 27 March 2006; accepted 11 April 2007
Available online 14 August 2007

**Abstract**

A warm standby $n$-system with operational and repair times following phase-type distributions is considered. The online unit goes through degradating levels, determined by inspections. Two types of repairs are performed, preventive and corrective, depending on the degradation level. The standby units undergo corrective repair. This systems is governed by a level-dependent-quasi-birth-and-death proces (LDQBD process), whose generator is constructed. The availability, rate of occurrence of failures, and other quantities of interest are calculated. A numerical example including an optimization problem and illustrating the calculations is presented. This system extend other previously studied in the literature.
© 2007 Elsevier B.V. All rights reserved.

*Keywords:* LDQBD process; Phase-type distribution; Inspection; Preventive and corrective repair; Degradation

## 1. Introduction

We study the maintenance of a multicomponent system under inspections, degradation, and different repair designs. Throughout the paper, the phase-type distributions play an important role. The versatility of these distributions will be shown throughout the paper. The class of these distributions is weakly dense in the family of distribution functions defined on the positive real line, so they allow to approach general lifetimes. When these distributions are involved, the procedures for solving the models are matrix-analytic methods. A book worth mentioning related to these methods is the one of Latouche and Ramaswami [1].

The literature about reliability systems is extensive. We reference papers studying systems related to matrix-analytic methods, then, the processes that govern the systems are generalized Markov processes. The classical text about this class of distributions is the one of Neuts [4]. Yeh [9] studied a system submitted to deterioration

* Corresponding author. Tel.: +34 958243155; fax: +34 958243267.
*E-mail address:* rperezo@ugr.es (R. Pérez-Ocón).

0377-2217/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2007.04.056

and inspections, calculating costs under optimal policies of these operations. A unit system under policy $N$ is studied in Neuts et al. [5] and Pérez-Ocón and Montoro-Cazorla [7]. In Pérez-Ocón and Montoro-Cazorla [6] a cold standby $n$-system is considered, and performance measures and the distribution of the up and down periods are calculated. The model that governs the system is a quasi-birth-and-death process (QBD process). This paper is extended in Pérez-Ocón and Montoro-Cazorla [8], where a warm standby $n$-system is considered, following the standby units exponential distributions. The model that governs this system is a level-dependent-quasi-birth-and-death process (LDQBD process). A two-system under degradation and with two repair modes is studied in Pérez-Ocón and Montoro-Cazorla [2].

We present a warm standby $n$-system, there is a online unit and the others are in standby or in repair. The operational level of the online unit is determined by random inspections. When the online fails one standby unit (if any) becomes the online one. There are good and bad operational levels, and preventive and corrective repairs. All repairs are as good as new. The lifetimes of the units and the repair times follow different continuous phase-type distributions, and the inspection times are governed by a discrete phase-type renewal process, because they occur in certain epochs.

For this system, the generator of the generalized Markov process that governs it is constructed, and it is an LDQBD process. The stationary probability vector, the availability, the rate of occurrence of failures for the different units in the system, and the costs are calculated. The distribution of the up and down periods are determined. A numerical application is performed, illustrating the calculations that have been considered throughout the paper, and calculating the optimal deteriorate level $g^*$, the threshold from which the preventive repair must be performed for maximizing the availability and costs. We extend the articles [6,8], where no structure of inspection is considered and degradation is not included.

The present paper suply a general model that extends other previously published in the literature. It is different to previous ones, as it includes all the following aspects: (1) The $n$-system includes degradation of the online unit; (2) the units undergo two types of repairs, preventive and corrective, depending on the degradation level of the online unit; (3) the inspections determine the degradation level of the online unit; (4) the preventive repair channel is interrupted for enlarging the lifetime of the system; (5) the system cost in terms of the costs of the involved operations is calculated.

The paper is organized as follows. In Section 2, we introduce the general Markov model. In Section 3, the stationary probability vector is calculated. The performance measures and costs are studied in Sections 4 and 5, respectively. In Section 6, the distributions of the up and down periods are determined. A numerical application and an optimizing problem are performed in Section 7. An Appendix is included where the construction of the generator is performed.

**Definition 1.** A continuous distribution $F(\cdot)$ on $[0,\infty)$ is a phase-type one with representation $(\alpha, T)$ if it is the distribution of the time until the absorption in a finite state Markov process with one absorbent state. Denoting by $\alpha$ the initial probability vector of the Markov process, the generator is given by

$$\begin{pmatrix} T & T^0 \\ 0 & 0 \end{pmatrix},$$

where $T$ denotes the rate transition matrix among the transient states and $T^0$ the column vector of absorption rates. The explicit expression of $F(\cdot)$ is $F(x) = 1 - \alpha\exp(Tx)e$, $x \geqslant 0$, $e$ being a column vector with all components equal to one. It will be denoted by $\mathrm{PH}(\alpha, T)$. The dimension of the matrix $T$ is the order of the distribution.

The discrete phase-type distributions are defined in a similar way, and they are denoted by $\mathrm{PH}_d(\alpha, T)$ where $T$ is the transition probability matrix among transient states and $T^0$ the column vector of absorption probabilities. For more details see Neuts [4].

**Definition 2.** If $A$ and $B$ are rectangular matrices of dimensions $m_1 \times m_2$ and $n_1 \times n_2$, respectively, their Kronecker product $A \otimes B$ is the matrix of dimensions $m_1n_1 \times m_2n_2$, written in compact form as $(a_{ij}B)$. The Kronecker sum of matrices $A$ and $B$ of orders $m$ and $n$, respectively, is defined by $A \oplus B = A \otimes I_n + I_m \otimes B$.

## 2. General model

We consider the following system with $n$ units:

1. There is a online unit and the others are in standby or in repair.
2. The level of the online unit is determined by random inspections. There are good and bad operational levels.
3. When the online fails one standby unit (if any) becomes the online one.
4. When the online unit occupies a bad level and an inspection occurs, it goes to preventive repair.
5. The corrective repair occurs when the online or one standby unit fails.
6. All repairs are as good as new.
7. If the only operational unit is the online one and if it fails, the unit being preventively repaired (if any) becomes the online unit even its repair had not been completed.
8. The system is up when there is one online unit, and it is down when all units are non-operational.

The units can be operational (online or standby) or non-operational (in repair, preventive or corrective, or waiting for repairing). It will be assumed that the online unit goes successively through the different degradating levels $\{1,\ldots,g,g+1,\ldots,m\}$ before the failure. The levels in $\{1,\ldots,g\}$ are good. If the unit occupies a level in $\{g+1,\ldots,m\}$ it goes to preventive repair, except if it is the only operational unit. If the unit fails, it goes to corrective repair. The operational and repair times have the following continuous distributions:

- The lifetime of the online unit follows a distribution $\mathrm{PH}(\alpha, T)$ of order $m$.
- The lifetime of the standby units are identical and follow a distribution $\mathrm{PH}(\alpha_s, T_s)$ of order $m_s$.
- The preventive repair follows a distribution $\mathrm{PH}(\beta_p, S_p)$ of order $n_p$.
- The corrective repair follows a distribution $\mathrm{PH}(\beta_c, S_c)$ of order $n$.
- The inspection times follows a discrete distribution $\mathrm{PH}_d(\gamma, L)$ of order $v$.
- If there are several units waiting for preventive repair and one of them is required to become instantaneously the online unit, it will be the one being repaired, and will start online with initial probability vector $\alpha_p$. This is a way to increase the lifetime of the system. The standby units undergo only corrective repairs.
- All the random times are independent.

### 2.1. State set

The states are defined in terms of the number of units in the different situations and the phases of the different distributions involved in the system. The state of the system at time $t$ can be identified by the following elements:

- the number of units in preventive repair, denoted by $i$, $i = 0,1,\ldots,n-1$,
- the number of units in corrective repair, denoted by $j$, $j = 0,1,\ldots,n$,
- the phase occupied by the online unit, denoted by $k$, $k = 1,2,\ldots,m$,
- the phases of the standby units, denoted by the vector $z^{(i,j)} = (z_1,\ldots,z_{n-1-(i+j)})$,
- the phase of the preventive repair, denoted by $l_p$, $l_p = 1,2,\ldots,n_p$,
- the phase of the corrective repair, denoted by $l_c$, $l_c = 1,2,\ldots,n_c$,
- the inspection phase, denoted by $f$, $f = 1,2,\ldots,v$.
- So, the following vector represents the exponential states of the system:

$$(i, j, k, z^{(i,j)}, l_p, l_c, f).$$

It is usual to group the states in sets determined by the way in which the system operates. If there are $i$ units in preventive repair and $j$ in corrective repair, in standby there are $n-1-(i+j)$ units. We have denoted by $z^{(i,j)}$ the vector whose entries are the phases occupied by the $n-1-(i+j)$ standby units, $z^{(i,j)} = (z_1,\ldots,z_{n-1-(i+j)})$, being $z_r$ the phase of the $r$th standby unit, with $1 \leqslant z_r \leqslant m_s$, $1 \leqslant r \leqslant n-1-(i+j)$.

Then, the possible values for the couple $(i,j)$ are

$$\{(i,j) : i = 0, 0 \leqslant j \leqslant n\} \cup \{(i,j) : 1 \leqslant i \leqslant n-1, 0 \leqslant j \leqslant n-i-1\}.$$

We give details about the phases of the model for different values of the couple $(i,j)$.

(1) No unit in repair: $(0,0) = \{(k, z^{(0,0)}, f) : 1 \leqslant k \leqslant m,\ 1 \leqslant f \leqslant v\}$.
(2) No unit in preventive repair:
  (a) with units in standby: $(0,j) = \{(k, z^{(0,j)}, l_c, f) : 1 \leqslant k \leqslant m,\ 1 \leqslant l_c \leqslant n_c,\ 1 \leqslant f \leqslant v\}, j = 1,\dots,n-2$,
  (b) without units in standby: $(0, n-1) = \{(k, l_c, f) : 1 \leqslant k \leqslant m,\ 1 \leqslant l_c \leqslant n_c,\ 1 \leqslant f \leqslant v\}$,
  (c) the system is down: $(0,n) = \{(l_c, f) : 1 \leqslant l_c \leqslant n_c,\ 1 \leqslant f \leqslant v\}$.

(3) No unit in corrective repair:
  (a) with units in standby: $(i,0) = \{(k, z^{(i,0)}, l_p, f) : 1 \leqslant k \leqslant m,\ 1 \leqslant l_p \leqslant n_p,\ 1 \leqslant f \leqslant v\}$, $i = 1,\dots,n-2$,
  (b) without units in standby: $(n-1, 0) = \{(k, l_p, f) : 1 \leqslant k \leqslant m,\ 1 \leqslant l_p \leqslant n_p,\ 1 \leqslant f \leqslant v\}$,

(4) Units in preventive and corrective repairs:
  (a) with units in standby: $(i,j) = \{(k, z^{(i,j)}, l_p, l_c, f) : 1 \leqslant k \leqslant m,\ 1 \leqslant l_p \leqslant n_p,\ 1 \leqslant l_c \leqslant n_c,\ 1 \leqslant f \leqslant v\}$, $i = 1,\dots,n-3, j = 1\dots,n-2-i$,
  (b) without units in standby: $(i,j) = \{(k, l_p, l_c, f) : 1 \leqslant k \leqslant m,\ 1 \leqslant l_p \leqslant n_p,\ 1 \leqslant l_c \leqslant n_c, 1 \leqslant f \leqslant v\}$, $i = 1,\dots,n-2, j = n-1-i$.

We remember that the system is down only when all the units are in corrective repair (repairing or waiting). On the other hand, the system cannot have all their units in preventive repair, in the instant of such occurrence, the unit under repair becomes online instantaneously.

## 2.2. Generator

We group the states of the system in macro-states. Define the macro-state $M$ as the number of non-operational units, $M$, $M = 0,\dots,n$. The non-operational units can be in repair preventive or corrective, then, the macro-state $n$ indicates that the system is down, and with the previous notation $M$ can be expressed as follows:

$$M = \{(i,j) : 0 \leqslant i \leqslant M,\ j = M - i\}, M = 0,\dots,n-1, \quad \text{and} \quad n = \{(0,n)\}.$$

The state space can be expressed as

$$S = \{M, M = 0, 1, \dots, n\}.$$

The generator is constructed in terms of the transitions among the macro-states. We calculate this generator by blocks. It can be observed that from a macro-state $M$ $(\neq 0, n)$ a transition is possible only to $M+1$ or $M-1$, since at every time $t$ only the failure of a unit or the completation of a repair is possible. Taking into account the possibilities in the boundary macro-states, the form of the generator is the following:

$$Q = \begin{pmatrix} B_{0,0} & B_{0,1} & & & & \\ B_{1,0} & A_1^{(1)} & A_0^{(1)} & & & \\ & A_2^{(2)} & A_1^{(2)} & A_0^{(2)} & & \\ & & \ddots & \ddots & \ddots & \\ & & & A_2^{(n-1)} & B_{n-1,n-1} & B_{n-1,n} \\ & & & & B_{n-1,n} & B_{n,n} \end{pmatrix}, \tag{2.1}$$

that corresponds to an LDQBD process. This expression is valid for $n \geqslant 4$ . The case $n = 4$ is the one with the minimum number of macro-states for which it is possible one unit in every defined situation: online, standby, in preventive or in corrective repair.

The structure of the degradation of the online unit is introduced by means of the following matrices:

$$U_1(g,m) \equiv U_1 = \begin{pmatrix} I_g & 0 \\ 0 & 0 \end{pmatrix}_{m\times m}, \quad U_2(g,m) \equiv U_2 = \begin{pmatrix} 0 & 0 \\ 0 & I_{m-g} \end{pmatrix}_{m\times m}. \tag{2.2}$$

In order to prepare the expressions for the calculation of transition rates, every macro-state $M = 2,\dots,n-1$ is partitioned in three sets corresponding to those with no preventive repair, with preventive and corrective repair, and with no preventive repair, denoted by $M_1, M_2, M_3$, respectively. We have

$$M = M_1 \cup M_2 \cup M_3$$

being

$$\begin{aligned} M_1 &= \{(0,M)\}, \\ M_2 &= \{(i,j) : 1 \leqslant i \leqslant M-1,\ j = M-i\}, \\ M_3 &= \{(M,0)\}. \end{aligned} \tag{2.3}$$

The macro-state $M = 0$ indicates that all units are operational, and it is formed by the couple $(0,0)$; the macro-state $M = 1$ is formed by the couples $\{(0,1),(1,0)\}$, and the macro-state $M = n$, is formed by the couple $(0,n)$ and indicates that the system is down. The macro-state 0 is included in $M_1$, the macro-state $n$ is included in $M_1$, and the macro-state 1 takes part of the macro-states $M_1$ and $M_3$. It will be seen throughout the paper that the macro-state 1 must be considered apart, as a border macro-state.

In advance, we denote by $(A \oplus \cdots \oplus A)_n$ the Kronecker sum of the matrix $A$ with itself n-times, and $I_c$ by the identity matrix of order $c$.

We express the blocks of the generator (2.1) in terms of the representations of the involved distributions in the system. The expressions of the blocks that form the generator and an explanation of their construction are given in Appendix.

## 3. Stationary probability vector

Once the generator is calculated, we determine the stationary probability vector, denoted by $\pi = (\pi_0, \pi_1, \dots, \pi_{n-1}, \pi_n)$. This vector satisfies the vectorial equations $\pi Q = 0$, $\pi e = 1$. It can be expressed in terms of the sets of macro-states (2.3) as follows:

$$\begin{aligned} \pi_0 &= (0,0) \quad \text{of order } m(m_s)^{n-1}v, \\ \pi_1 &= \{(0,1),(1,0)\} \quad \text{of order } m(m_s)^{n-2}n_c v + m(m_s)^{n-1}n_p v, \\ \pi_n &= (0,n) \quad \text{of order } n_c v \end{aligned}$$

and

$$\pi_M = (\pi_M(1), \pi_M(2), \pi_M(3)), M = 2,\dots,n-1 \quad \text{of order } (m_s)^{n-1-M}n_c v + (m_s)^{n-1-M}n_p n_c v + (m_s)^{n-1-M}n_p v.$$

The calculations that follow have been established in Pérez-Ocón and Montoro-Cazorla [8, Section 3] step by step and there it was seen that the solution of the system is

$$\pi_j = \pi_0 \prod_{i=0}^{j-1} R_i, \quad j = 1,\dots,n, \tag{3.1}$$

where $\pi_0$ is determined from the matricial equation

$$\pi_0(B_{0,0} + R_0 B_{1,0}) = \mathbf{0},$$

subject to the normalization condition

$$\pi_0\left(\sum_{j=0}^{n}\prod_{i=0}^{j-1} R_i\right)e = 1.$$

The matrices $R_0, R_1, R_2, \ldots, R_{n-2}$ involved in the above expressions are given by

$$\begin{aligned}
R_{n-1} &= -B_{n-1,n}B_{n,n}^{-1}, \\
R_{n-2} &= -A_0^{(n-2)}(B_{n-1,n-1} + R_{n-1}B_{n,n-1})^{-1}, \\
R_{j-1} &= -A_0^{(j-1)}\left(A_1^{(j)} + R_jA_2^{(j+1)}\right)^{-1}, \quad j = n-2, \ldots, 2, \\
R_0 &= -B_{0,1}(A_1^{(1)} + R_1A_2^{(2)})^{-1}
\end{aligned} \tag{3.2}$$

being the matrices $(A_1^{(j)} + R_jA_2^{(j+1)})$ non-singular for $j = 2, \ldots, n-2$.

For getting these formulae we use previous results of Naoumov [3, Proposition 18].

## 4. Performance measures

In this section, we calculate the availability and the mean number of failures per unit time of the units and system. Also, we calculate the mean number of the inspections and the times that the preventive repair channel is interrupted per unit time.

### 4.1. Availability

The probability that the system will be operational at time $t$ is the probability that the system does not occupy the down state $n$, so we have

$$A = \sum_{i=0}^{n-1} \pi_i e = \pi_0\left(\sum_{i=0}^{n-1}\prod_{j=0}^{i-1} R_j\right)e = 1 - \pi_n e. \tag{4.1}$$

### 4.2. Rate of occurrence of failures

This measure is the mean number of failures per unit time, and it is known in the literature as the rocof. We calculate the rocof of the corrective repairs for the units and the system. For calculating the different rocofs we consider three cases, corresponding to $M = 0$, $M = 1$, and the others. We express the final expressions in terms of the matrices in (3.2). For calculating the rocof, we must determine the macro-states from which the fail is produced, and the probability of occupancy of these macro-states. This measure has been calculated when phase-type distributions are involved in [5].

#### 4.2.1. Online unit

If the online unit is in macro-state 0, the vector of failure rates is

$$C_0 = T^0 \otimes e_{(m_s)^{n-1}v}.$$

If $M = 1$, the failure can occur from $(0,1)$ and $(1,0)$ and the rates are given in the vector

$$C_1 = \begin{pmatrix} T^0 \otimes e_{(m_s)^{n-2}n_cv} \\ T^0 \otimes e_{(m_s)^{n-2}n_pv} \end{pmatrix}.$$

The vector of failure rates in the other cases is a three-block vector, corresponding to the sets in (2.3), with intermediate blocks identical given by

$$C_k = \begin{pmatrix} T^0 \otimes e_{(m_s)^{n-1-k}n_cv} \\ e_{k-1} \otimes T^0 \otimes e_{(m_s)^{n-1-k}n_pn_cv} \\ T^0 \otimes e_{(m_s)^{n-1-k}n_pv} \end{pmatrix}; \quad k = 2, \ldots, n-1.$$

Finally, the rate of occurrence of failures of the online unit is

$$v_1 = \pi_0 C_0 + \sum_{k=1}^{n-1} \pi_k C_k = \pi_0 C_0 + \pi_0 \left( \sum_{k=1}^{n-1} \prod_{j=0}^{k-1} R_j \right) C_k. \tag{4.2}$$

### 4.2.2. Standby units

The procedure is similar to the calculation of the rocof for the online unit. For $M = 0$ the failure rates vector is

$$D_0 = (I_m \otimes (T_s^0 \oplus \cdots \oplus T_s^0)_{n-1} \otimes \beta_c \otimes I_v) e_{m(m_s)^{n-2} n_c v}.$$

For $M = 1$, the failure can occur from $(0,1)$ and $(1,0)$ and the rates are given in the vector

$$D_1 = \begin{pmatrix} (I_m \otimes (T_s^0 \oplus \cdots \oplus T_s^0)_{n-2} \otimes I_{n_c v}) e_{m(m_s)^{n-3} n_c v} \\ (I_m \otimes (T_s^0 \oplus \cdots \oplus T_s^0)_{n-2} \otimes I_{n_p} \otimes \beta_c \otimes I_v) e_{m(m_s)^{n-3} n_p n_c v} \end{pmatrix}.$$

The vector of failure rates in the other cases is a three-block vector, corresponding to the sets (2.3), with intermediate blocks identical given by

$$D_k = \begin{pmatrix} (I_m \otimes (T_s^0 \oplus \cdots \oplus T_s^0)_{n-1-k} \otimes I_{n_c v}) e_{m(m_s)^{n-2-k} n_c v} \\ (e_{k-1} \otimes I_m \otimes (T_s^0 \oplus \cdots \oplus T_s^0)_{n-1-k} \otimes I_{n_p n_c v}) e_{m(m_s)^{n-2-k} n_p n_c v} \\ (I_m \otimes (T_s^0 \oplus \cdots \oplus T_s^0)_{n-1-k} \otimes I_{n_p} \otimes \beta_c \otimes I_v) e_{m(m_s)^{n-2-k} n_p n_c v} \end{pmatrix}$$

for $k = 2, \ldots, n-2$.

Then, the rate of occurrence of failures of the standby units is

$$v_2 = \sum_{k=0}^{n-2} \pi_k D_k = \pi_0 \sum_{k=0}^{n-2} \left( \prod_{j=0}^{k-1} R_j \right) D_k. \tag{4.3}$$

### 4.2.3. The system

The system fails when there are $n-1$ units in corrective repair and the online unit fails. The only macro-states from which the failure is possible are the ones $(0, n-1)$ in $M_1$. Then, the matrix that represents the transition rates is

$$E = \begin{pmatrix} T^0 \otimes e_{n_c v} \\ 0 \\ 0 \end{pmatrix}.$$

Thus, the rocof of the system is

$$v_3 = \pi_{n-1} E = \pi_0 \left( \prod_{j=0}^{n-2} R_j \right) E. \tag{4.4}$$

## 4.3. Mean number of interrupted preventive repairs per unit time

This operation of interruption of the preventive repair occurs in the evolution of the system, and we measure its occurrence by means of this quantity. This operation is performed for the system will be operational the most time possible. An interruption occurs when there is only one operational unit (online) and it undergoes a failure. Then, the unit being preventively repaired becomes the online unit. The system must occupy the macro-state $n-1$ and at least one unit has to be in preventive repair. Then, the non-null rates are the ones of the elements $(1, n-2), \ldots, (n-1, 0)$, and the vector of rates is

$$F = \begin{pmatrix} 0 \\ e_{n-2} \otimes T^0 \otimes e_{n_p n_c v} \\ T^0 \otimes e_{n_p v} \end{pmatrix}.$$

The mean number of interrupted repair per unit time is

$$v_4 = \pi_{n-1} F = \pi_0 \left( \prod_{j=0}^{n-2} R_j \right) F. \tag{4.5}$$

### 4.4. Rate of occurrence of inspections

Now we calculate the mean number of inspections per unit time. An inspection arrives governed by the vector $L^0$. Any other change among the phases of the operations in the system cannot occur. If $M = 0$, the vector of transition rates is given by $G_0 = e_{m(m_s)^{n-1}} \otimes L^0$. If $M = 1 = \{(0,1),(1,0)\}$, the vector is

$$G_1 = \begin{pmatrix} e_{m(m_s)^{n-2} n_c} \otimes L^0 \\ e_{m(m_s)^{n-2} n_p} \otimes L^0 \end{pmatrix}.$$

For $k = 2, \ldots, n-1$, the vector of transition rates is

$$G_k = \begin{pmatrix} e_{m(m_s)^{n-1-k} n_c} \otimes L^0 \\ e_{k-1} \otimes e_{m(m_s)^{n-1-k} n_p n_c} \otimes L^0 \\ e_{m(m_s)^{n-1-k} n_p} \otimes L^0 \end{pmatrix}, \quad k = 2, \ldots, n-1.$$

For $M = n$, we have $G_n = e_{n_c} \otimes L^0$. Then, the final expression is

$$v_5 = \sum_{k=0}^{n} \pi_k G_k = \pi_0 \sum_{k=0}^{n} \left( \prod_{j=0}^{k-1} R_j \right) G_k. \tag{4.6}$$

## 5. Costs

We introduce the study of the generated costs of the system per unit time in terms of the costs of the different situations involved in the system. This is a complex system, and every operation implies a cost. The positive cost is generated when the system is operational, and the negative costs are due to the repairs (preventive and corrective), the inspections, and the interrumpted preventive repair. We denoted these costs per unit time by

| | |
|---|---|
| $c_u$ | operational cost |
| $c_{rp}$ | cost due to preventive repair |
| $c_{rc}$ | cost due to corrective repair |
| $c_I$ | cost due to inspection |
| $c_{rpi}$ | cost due to the interruption of preventive repair |

The total cost of the system per unit time is denoted by $c$.

First, we calculate the generated costs by repairs. If the system occupies the macro-state 0, it gets benefits. We represent these benefits by a column vector with the order of the vector $\pi_0$. We have $H_0 = c_u e$.

If the system occupies the set $M_1 = \{(0,1),(1,0)\}$, the costs due to the preventive or corrective repair included in these macro-states must be incorporated. Ordering these macro-states we define the column vector

$$H_1 = \begin{pmatrix} (c_u + c_{rc}) e \\ (c_u + c_{rp}) e \end{pmatrix},$$

where the vectors $e$ have the orders or $(0,1)$ and $(1,0)$, respectively.

If the system occupies the macro-state $n$, the only costs are produced by the corrective repair. We consider the vector $H_n = c_{rc}e$, with order the one of $\pi_n$.

For the rest of macro-states we must consider the sets in (2.3). The resultant vector is

$$H_k = \begin{pmatrix} (c_u + c_{rc})e \\ (c_u + c_{rp} + c_{rc})e \\ (c_u + c_{rp})e \end{pmatrix}; \quad k = 2, \ldots, n-1$$

being the costs in the rows due to the produced costs in the macro-states of $M_1$, $M_2$, and $M_3$, respectively. The order of the vectors $e$ are the appropriate for the vector $H_k$ will have the order of $\pi_k$.

The costs due to interrupted preventive repairs and to inspections must be added. As the costs are per unit time, for calculating these costs we need the mean number of these events per unit time, that have been calculated in (4.5) and (4.6). Finally, we have the total cost of the system per unit time:

$$c = \sum_{k=0}^{n} \pi_k H_k + v_4 c_{rpi} + v_5 c_I.$$

## 6. Distribution functions for the up and down periods

These distribution functions have been calculated in the previous paper of Pérez-Ocón and Montoro-Cazorla [2] for a simpler system. In the present one, we have added wear, inspections and repairs. The up and down periods follow PH-distributions whose representations are determined below.

The up period is the timespan between the instant at which all the units are initially operational and the instant at which all the units are not operational for the first time. We consider a modified Markov process from the original, with the same operational macro-states and identifying the non-operational macro-states in a new absorbent macro-state that will be denoted by $n^*$. The up period is the time up to the absorption by the macro-state $n^*$, and thus the distribution will be a PH-distribution. The generator of this new Markov process is

$$Q^* = \left( \begin{array}{|cccccc|c} B_{00} & B_{01} & & & & & 0 \\ B_{10} & A_1^{(1)} & A_0^{(1)} & & & & 0 \\ & A_2^{(2)} & A_1^{(2)} & A_0^{(2)} & & & 0 \\ & & \ddots & \ddots & \ddots & & 0 \\ & & & A_2^{(n-1)} & B_{n-1,n-1} & & B^*_{n-1,n} \\ \hline 0 & 0 & 0 & 0 & 0 & & 0 \end{array} \right) \tag{6.1}$$

being $B^*_{n-1,n} = B_{n-1,n}e$.

The representation of the up period is $(\phi_u, Q*)$. The initial conditions need to be chosen so as to reflect the physical conditions of the system at time $t = 0$. For example, if all units are new at $t = 0$, the initial vector can be chosen as

$$\phi_u = \left[ \frac{\pi_0}{\pi_0 e}, \mathbf{0} \right]. \tag{6.2}$$

The operational mean time is

$$\text{MTTF} = -\phi_u \Lambda_u^{-1} e. \tag{6.3}$$

The down period begins when the only operational unit fails (the rest are in repair or waiting for repair), and finishes at the point when the first repair is completed. This period follows a PH$(\phi_d, \Lambda_d)$, where $\phi_d$ is determined taking into account that the macro-states have been grouped in the three sets in (2.3). Then, we write $\pi_{n-1} = (\pi_{n-1}(1), \pi_{n-1}(2), \pi_{n-1}(3))$.

We denote by $\pi_{n-1}^{(i,j,k)}(1)$ the probability that, at time $t$, in the system there are $n-1$ units in corrective repair, the online unit occupies the phase $i$, the unit being repaired is in phase $j$, and the inspection is in phase $k$, with $1 \leqslant i \leqslant m$, $1 \leqslant j \leqslant m_s$, $1 \leqslant k \leqslant v$.

We denote by $\phi_d(j,k)$ the probability that the system fails when the unit under repair is in phase $j$ and the inspection in phase $k$, $1 \leqslant j \leqslant m_s$, $1 \leqslant k \leqslant v$. This probability can be expressed as

$$\phi_d(j,k) = \frac{\sum_{i=1}^{m} \pi_{n-1}^{(i,j,k)}(1) T_j^0}{\pi_{n-1}(1)(T^0 \otimes e_{n_c v})}, \quad 1 \leqslant j \leqslant m_s, \quad 1 \leqslant k \leqslant v,$$

$T_j^0$ being the $j$th entries of the column vector $T^0$.

The initial vector is

$$\phi_d = (\phi_d(1,1), \ldots, \phi_d(1,v), \ldots, \phi_d(n_c,1), \ldots, \phi_d(n_c,v))$$

and the matrix $\Lambda_d$ is

$$\Lambda_d = S_c \oplus (L + L^0\gamma).$$

The mean time that the system is no-operational is

$$\mathrm{MTTD} = -\phi_d \Lambda_d^{-1} e. \tag{6.4}$$

## 7. Numerical application

We illustrate the general model by means of an example. We consider five units ($n = 5$) with the following distributions and parameters.

Online unit: PH($\alpha, T$) with

$$\alpha = (1,0,0,0,0), \quad \alpha_p = (0.1,0.3,0.2,0.3,0.1), \quad T = \begin{pmatrix} -0.0081 & 0.0081 & 0 & 0 & 0 \\ 0 & -0.0240 & 0.0240 & 0 & 0 \\ 0 & 0 & -0.009 & 0.009 & 0 \\ 0 & 0 & 0 & -0.0072 & 0.0072 \\ 0 & 0 & 0 & 0 & -0.084 \end{pmatrix}$$

being $\alpha_p$ the initial probability vector for the online unit when this is the one interrupted in the preventive repair because there are no standby unit.

Standby units: PH($\alpha_s, T_s$) with

$$\alpha_s = (1,0), \quad T_s = \begin{pmatrix} -0.0052 & 0.0052 \\ 0.0013 & -0.0052 \end{pmatrix}.$$

Preventive repair times: PH($\beta_p, S_p$) with

$$\beta_p = (1,0), \quad S_p = \begin{pmatrix} -0.0667 & 0.0667 \\ 0 & -0.0667 \end{pmatrix}.$$

Corrective repair times: PH($\beta_c, S_c$) with

$$\beta_c = (1,0), \quad S_c = \begin{pmatrix} -0.02 & 0.02 & 0 \\ 0.01 & -0.08 & 0.07 \\ 0.005 & 0 & -0.10 \end{pmatrix}.$$

Inspection time: PH($\gamma, L$) with

$$\gamma = (1,0), \quad L = \begin{pmatrix} -0.02 & 0.02 \\ 0 & -0.02 \end{pmatrix}.$$

We remember that $g$ is the number of good states in the online unit. We present in Table 1 the performance measures and costs calculated in Sections 4 and 5 in terms of the values of $g$.

It is noted that when $g$ increases the availability and the costs decreases. So the value $g = 1$ makes these quantities maximum. The preventive repair will begin in the level 2 of the online unit.

Table 1
Performance measures and costs for different values of $g$

| $g$ | $A$ | $v_1$ | $v_2$ | $v_3$ | $v_4$ | $v_5$ | $c$ |
|---|---|---|---|---|---|---|---|
| 1 | 0.9999 | $(4.0447)^{-004}$ | 0.0023 | $(1.9460)^{-006}$ | $(3.8147)^{-008}$ | 0.01 | 0.9016 |
| 2 | 0.9995 | $(5.3417)^{-004}$ | 0.0059 | $(8.0364)^{-006}$ | $(8.6752)^{-008}$ | 0.01 | 0.7987 |
| 3 | 0.9990 | $(9.5882)^{-004}$ | 0.0064 | $(1.5609)^{-005}$ | $(5.8788)^{-008}$ | 0.01 | 0.7911 |
| 4 | 0.9979 | 0.0021 | 0.0074 | $(3.2178)^{-005}$ | $(8.9212)^{-009}$ | 0.01 | 0.7709 |

## 8. Conclusion

We have considered a complex system under different operations of reliability. We show that using matrix-analytic methods it is possible to present expressions that can be computationally implemented, and consequently applicable. In spite of the complexity of the system, it can be seen throughout the paper, and particularly constructing the generator of the general model, that in the blocks forming this matrix there is a repetitive structure. This is an interesting consequence of using the matrix-analytic methods. Our future research on modelization of systems will be related to find phase-type distributions with properties of interest from the point of view of reliability, such as bathtub and upbathtub shaped, and to approach the models we construct to practical structures, supplying new procedures to the engineers and applied probabilistician.

## Acknowledgement

This paper is partially supported by the Ministerio de Educación y Ciencia, España, under the grant MTM2004-03672.

## Appendix A. Construction of the generator

We construct the blocks that form the infinitesimal generator. Some indications are given for constructing certain blocks. The others can be constructed following the previous indications, since many situations are repeated. The subindices indicate the order of the matrices and vectors.

### *A.1. Initial boundary blocks*

#### *A.1.1. Transition 0 → 0*

The block $B_{00}$ represents the rate transition among phases when there is no change among macro-states. The expression of this block $B_{00}$ in terms of the involved distribution functions is

$$B_{0,0} = T \otimes I_{(m_s)^{n-1}v} + U_1 \otimes I_{(m_s)^{n-1}} \otimes L^0\gamma + I_m \otimes (T_s \oplus \cdots \oplus T_s)_{n-1} \otimes I_v + I_{m(m_s)^{n-1}} \otimes L.$$

The interpretation of this expression is the following. The first summand indicates that the unit online change among operational phases, the standby units do not change of phases, and there is no inspection; the second one that there is an inspection and it finds the online unit is in a good level, there is no change among the standby units; the third one indicates that the online unit does not change of phase, some of the standby units change of phase, and the inspection does not change; and the fourth one indicates that there is no change among the phases of all units, and there is a change of phase in the inspection time.

#### *A.1.2. Transition 0 → 1*

The block $B_{01}$ represents the rate transition block when all units are operational, and there is a failure, that can need preventive or corrective repair. The following transition table:

| | $(0,1)$ | $(1,0)$ |
|---|---|---|
| $(0,0)$ | $B_{0,1}(1)$ | $B_{0,1}(2)$ |

shows that $B_{0,1}=[B_{0,1}(1),B_{0,1}(2)]$, being $B_{0,1}(1)$ the block corresponding to a corrective failure in the online or in one of the standby units, and $B_{0,1}(2)$, the one corresponding to a failure with preventive repair in the online unit. In this last case, an inspection that has found the online unit in a bad level has occurred, then a standby unit becomes online unit. We have

$$\begin{aligned}
B_{0,1}(1) &= T^0\alpha\otimes e_{m_s}\otimes I_{(m_s)^{n-2}}\otimes\beta_c\otimes I_v + I_m\otimes(T_s^0\oplus\cdots\oplus T_s^0)_{n-1}\otimes\beta_c\otimes I_v,\\
B_{0,1}(2) &= U_2 e_m\alpha\otimes e_{m_s}\otimes I_{(m_s)^{n-2}}\otimes\beta_p\otimes L^0\gamma.
\end{aligned}$$

### A.1.3. Transition 1 → 0

The block $B_{1,0}$ represents the transition when a repair, preventive or corrective, is completed. The following transition table:

| | (0, 0) |
|---|---|
| (0, 1) | $B_{1,0}(1)$ |
| (1, 0) | $B_{1,0}(2)$ |

shows that

$$B_{1,0}=\begin{bmatrix}B_{1,0}(1)\\ B_{1,0}(2)\end{bmatrix},$$

where $B_{1,0}(1)$ corresponds to the completion of a corrective repair and $B_{1,0}(2)$ to the one of a preventive repair. We have

$$\begin{aligned}
B_{1,0}(1) &= I_{m(m_s)^{n-2}}\otimes\alpha_s\otimes S_c^0\otimes I_v,\\
B_{1,0}(2) &= I_{m(m_s)^{n-2}}\otimes\alpha_s\otimes S_p^0\otimes I_v.
\end{aligned}$$

## A.2. Diagonal blocks

### A.2.1. Blocks $A_1^{(M)}$

These correspond to the transition $M\to M$ ($M=1,\ldots,n-2$). They are the blocks denoted by $A_1^{(M)}$ in (2.1). They represent that there is no failure among the units, only change among the phases occurs. The blocks can be written as

$$A_1^{(M)} = \begin{array}{c} \\ M_1\\ M_2\\ M_3\end{array}\begin{array}{c}\begin{array}{ccc} M_1 & M_2 & M_3\end{array}\\ \begin{pmatrix} A_1^{(M)}(1,1) & & \\ & A_1^{(M)}(2,2) & \\ & & A_1^{(M)}(3,3)\end{pmatrix}\end{array},\quad M=2,\ldots,n-2,$$

where

$$\begin{aligned}
A_1^{(M)}(1,1) &= T\otimes I_{(m_s)^{n-M-1}n_cv} + I_m\otimes(T_s\oplus\cdots\oplus T_s)_{n-M-1}\otimes I_{n_cv} + I_{m(m_s)^{n-M-1}}\otimes S_c\otimes I_v\\
&\quad + I_{m(m_s)^{n-M-1}n_c}\otimes L + U_1\otimes I_{(m_s)^{n-M-1}n_c}\otimes L^0\gamma,\\
A_1^{(M)}(2,2) &= I_{M-1}\otimes\Big\{T\otimes I_{(m_s)^{n-M-1}n_pn_cv} + I_m\otimes(T_s\oplus\cdots\oplus T_s)_{n-M-1}\otimes I_{n_pn_cv}\\
&\quad + I_{m(m_s)^{n-M-1}}\otimes S_p\otimes I_{n_cv} + I_{m(m_s)^{n-M-1}n_p}\otimes S_c\otimes I_v\\
&\quad + I_{m(m_s)^{n-M-1}n_pn_c}\otimes L + U_1\otimes I_{(m_s)^{n-M-1}n_pn_c}\otimes L^0\gamma\Big\},\\
A_1^{(M)}(3,3) &= T\otimes I_{(m_s)^{n-M-1}n_pv} + I_m\otimes(T_s\oplus\cdots\oplus T_s)_{n-M-1}\otimes I_{n_pv} + I_{m(m_s)^{n-M-1}}\otimes S_p\otimes I_v\\
&\quad + I_{m(m_s)^{n-M-1}n_p}\otimes L + U_1\otimes I_{(m_s)^{n-M-1}n_p}\otimes L^0\gamma.
\end{aligned}$$

The block $A_1^{(M)}(1,1)$ has five summands, corresponding to the change of phase of the following: the online; one of the standby units; the corrective repair; the inspection; and there is an inspection when the online is in a good level. When there is a change of phase in one of these, the others do not change.

The case $M=1$ must be considered apart, as we have seen in (2.3), and the sets $M_1$ and $M_3$ are included. We have

$$A_1^{(1)} = \begin{bmatrix} A_1^{(1)}(1,1) & \\ & A_1^{(1)}(3,3) \end{bmatrix}.$$

The blocks $A_1^{(1)}(1,1)$ and $A_1^{(1)}(3,3)$ have the above expressions for $M=1$. So, we have justified $A_1^{(M)}$ for $M=1,\dots,n-2$. The matrix $A_1^{(M)}(2,2)$ represent the changes among the elements of $M_2$, what means that there is only change among the phases, and from the couple $(i,j)$ in $M_2$ only can pass to the couple $(i,j)$, this is represented by the unit matrix $I_{M-1}$. The entries of the matrices represent change among the phases of the five elements involved, and have similar expressions to the previous ones.

### A.2.2. Blocks $A_0^{(M)}$

These correspond to the transition $M \to M+1, M=1,2,\dots,n-2$. These blocks represent that there is a failure in the system, preventive or corrective:

$$A_0^{(M)} = \begin{bmatrix} A_0^{(M)}(1,1) & A_0^{(M)}(1,2) & \\ & A_0^{(M)}(2,2) & \\ & A_0^{(M)}(3,2) & A_0^{(M)}(3,3) \end{bmatrix}, \quad M=2,\dots,n-2$$

and

$$A_0^{(1)} = \begin{bmatrix} A_0^{(1)}(1,1) & A_0^{(1)}(1,2) & \\ & A_0^{(1)}(3,2) & A_0^{(1)}(3,3) \end{bmatrix}$$

being for $M=1,\dots,n-2$:

$$A_0^{(M)}(1,1) = T^0\alpha \otimes e_{m_s} \otimes I_{(m_s)^{n-M-2}n_cv} + I_m \otimes (T_s^0 \oplus \cdots \oplus T_s^0)_{n-M-1} \otimes I_{n_cv},$$

$$A_0^{(M)}(1,2) = \left[U_2 e_m\alpha \otimes e_{m_s} \otimes I_{(m_s)^{n-M-2}} \otimes \beta_p \otimes I_{n_c} \otimes L^0\gamma, \mathbf{0}\right],$$

$$A_0^{(M)}(2,2) = \begin{pmatrix} A_0^{(M),(1,1)}(2,2) & A_0^{(M),(1,2)}(2,2) & & \\ & \ddots & \ddots & \\ & & A_0^{(M),(M-1,M-1)}(2,2) & A_0^{(M),(M-1,M)}(2,2) \end{pmatrix},$$

$$A_0^{(M),(i,i)}(2,2) = T^0\alpha \otimes e_{m_s} \otimes I_{(m_s)^{n-M-2}n_pn_cv} + I_m \otimes (T_s^0 \oplus \cdots \oplus T_s^0)_{n-M-1} \otimes I_{n_pn_cv}; \quad i=1,\dots,M-1,$$

$$A_0^{(M),(i,i+1)}(2,2) = U_2 e_m\alpha \otimes e_{m_s} \otimes I_{(m_s)^{n-M-2}n_pn_c} \otimes L^0\gamma; \quad i=1,\dots,M-1,$$

$$A_0^{(M)}(3,2) = \left[\mathbf{0}, T^0\alpha \otimes e_{m_s} \otimes I_{(m_s)^{n-M-2}n_p} \otimes \beta_c \otimes I_v + I_m \otimes (T_s^0 \oplus \cdots \oplus T_s^0)_{n-M-1} \otimes I_{n_p} \otimes \beta_c \otimes I_v\right],$$

$$A_0^{(M)}(3,3) = U_2 e_m\alpha \otimes e_{m_s} \otimes I_{(m_s)^{n-M-2}} \otimes I_{n_p} \otimes L^0\gamma.$$

The only blocks that need to be explained are the $A_0^{(M)}(1,2)$, $A_0^{(M)}(3,2)$, and $A_0^{(M)}(2,2)$, the others are similar to previous ones. We reason on a particular case $M=2$, then the block $A_0^{(2)}(1,2)$ represents the transition between $M_1=(0,2)$ and $M_2=\{(1,2),(2,1)\}$. It is clear that the transition $(0,2)\to(1,2)$ means that a failure with preventive repair occurs, and the transition $(0,2)\to(2,1)$ is not possible. This justify the above expression. With the block $A_0^{(M)}(3,2)$ the situation is similar. For constructing the block $A_0^{(M)}(2,2)$ we must consider that the couples $(i,j)$ in $M_2$ only can pass to the couples $(i+1,j)$ or $(i,j+1)$ when a failure arrives, being the other possibilities null. Ordering lexicographically the couples the final form of the matrix $A_0^{(M)}(2,2)$ is determined.

### A.2.3. Blocks $A_2^{(M)}$

These blocks correspond to the transition $M \to M-1$, $M = 1,\ldots,n-1$ and they represent the completation of a preventive or corrective repair. Considering the sets (2.3), the matrix corresponding to this transitions can be written as

$$A_2^{(M)} = \begin{bmatrix} A_2^{(M)}(1,1) & & \\ A_2^{(M)}(2,1) & A_2^{(M)}(2,2) & A_2^{(M)}(2,3) \\ & & A_2^{(M)}(3,3) \end{bmatrix}, \quad M = 3,\ldots,n-1,$$

$$A_2^{(2)} = \begin{bmatrix} A_2^{(2)}(1,1) & \\ A_2^{(2)}(2,1) & A_2^{(2)}(2,3) \\ & A_2^{(2)}(3,3) \end{bmatrix}$$

being

$$A_2^{(M)}(1,1) = I_{m(m_s)^{n-M-1}} \otimes \alpha_s \otimes S_c^0\beta_c \otimes I_v,$$

$$A_2^{(M)}(2,1) = \begin{bmatrix} I_{m(m_s)^{n-M-1}} \otimes \alpha_s \otimes S_p^0 \otimes I_{n_c v} \\ \mathbf{0} \end{bmatrix}$$

$$A_2^{(M)}(2,2) = \left(\begin{array}{c|cccc} & 1 & 2 & \cdots & M-2 \\ 1 & A_2^{(M),(1,1)}(2,2) & & & \\ 2 & A_2^{(M),(2,1)}(2,2) & \ddots & & \\ \vdots & & \ddots & & \\ M-2 & & & & A_2^{(M),(M-2,M-2)}(2,2) \\ M-1 & & & & A_2^{(M),(M-1,M-2)}(2,2) \end{array}\right),$$

$$A_2^{(I),(i,i)}(2,2) = I_{m(m_s)^{n-I-1}} \otimes \alpha_s \otimes I_{n_p} \otimes S_c^0\beta_c \otimes I_v; \quad i = 1,\ldots I-1,$$

$$A_2^{(I),(i,i-1)}(2,2) = I_{m(m_s)^{n-I-1}} \otimes \alpha_s \otimes S_p^0\beta_p \otimes I_{n_c v}; \quad i = 1,\ldots I-1,$$

$$A_2^{(I)}(2,3) = \begin{bmatrix} \mathbf{0} \\ I_{m(m_s)^{n-I-1}} \otimes \alpha_s \otimes I_{n_p} \otimes S_c^0 \otimes I_v \end{bmatrix},$$

$$A_2^{(I)}(3,3) = I_{m(m_s)^{n-I-1}} \otimes \alpha_s \otimes S_p^0\beta_p \otimes I_v.$$

These matrices are justified reasoning as in the blocks $A_0^{(M)}$.

### A.3. Final boundary blocks

These matrices can be constructed in a similar way to the previous ones.

$$B_{n-1,n-1} = \begin{bmatrix} B_{n-1,n-1}(1,1) & & \\ B_{n-1,n-1}(2,1) & B_{n-1,n-1}(2,2) & \\ & B_{n-1,n-1}(3,2) & B_{n-1,n-1}(3,3) \end{bmatrix}$$

being

$$\begin{aligned}
B_{n-1,n-1}(1,1) &= T \otimes I_{n_c v} + I_m \otimes S_c \otimes I_v + I_{mn_c} \otimes L + I_{mn_c} \otimes L^0\gamma \\
&= (T \oplus S_c) \oplus (L + L^0\gamma), \\
B_{n-1,n-1}(2,1) &= \begin{bmatrix} T^0\alpha_p \otimes e_{n_p} \otimes I_{n_c v} \\ \mathbf{0} \end{bmatrix},
\end{aligned}$$

$$B_{n-1,n-1}(2,2) =$$

$$\begin{array}{c|c}
 & \begin{array}{cccc} \mathbf{1} & \mathbf{2} & \ldots\ldots & \mathbf{n-2} \end{array} \\
\begin{array}{c} \mathbf{1} \\ \mathbf{2} \\ \\ \mathbf{n-2} \end{array} & \begin{pmatrix} B^{(1,1)}_{n-1,n-1}(2,2) & & & \\ B^{(2,1)}_{n-1,n-1}(2,2) & B^{(2,2)}_{n-1,n-1}(2,2) & & \\ & \ddots & \ddots & \\ & & B^{(n-2,n-3)}_{n-1,n-1}(2,2) & B^{(n-2,n-2)}_{n-1,n-1}(2,2) \end{pmatrix}
\end{array},$$

and

$$\begin{aligned}
B^{(i,i)}_{n-1,n-1}(2,2) &= T \otimes I_{n_p n_c v} + I_m \otimes S_p \otimes I_{n_c v} + I_{mn_p} \otimes S_c \otimes I_v \\
&\quad + I_{mn_p n_c} \otimes (L + L^0\gamma); \quad i = 1, \ldots, M-1, \\
B^{(i,i-1)}_{n-1,n-1}(2,2) &= T^0\alpha_p \otimes e_{n_p}\beta_p \otimes I_{n_c v}; \quad i = 2, \ldots, I-1, \\
B_{n-1,n-1}(3,2) &= \left[\mathbf{0}, T^0\alpha_p \otimes e_{n_p}\beta_p \otimes \beta_c \otimes I_v\right], \\
B_{n-1,n-1}(3,3) &= T \otimes I_{n_p v} + I_m \otimes S_p \otimes I_v + I_{mn_p} \otimes (L + L^0\gamma), \\
B_{n-1,n} &= \begin{bmatrix} T^0 \otimes I_{n_c v} \\ \mathbf{0} \end{bmatrix}, \\
B_{n,n-1} &= [\alpha \otimes S_c^0\beta_c \otimes I_v, \mathbf{0}], \\
B_{n,n} &= S_c \oplus (L + L^0\gamma).
\end{aligned}$$

These matrices can be analyzed as the previous ones. We explain only the matrix the blocks $B^{(i,i-1)}_{n-1,n-1}(2,2)$ and $B_{n-1,n-1}(3,2)$, because they are different to the others previously considered. The block $B_{n-1,n-1}(3,2)$ represents the transition rate between the elements $(n-1,0) \to (n-2,1)$ in $M_2$, all the others transition rates $(n-1,0) \to (n-j-1,j), j \neq 0$ being null, and the non-null transition rate means that the online unit undergoes a failure needing corrective repair and the unit being served in preventive repair becomes the online unit. The block $B^{(i,i-1)}_{n-1,n-1}(2,2)$ represents the changes among the elements of $\mathrm{M}_2$ when there is only change of phase. But by the assumptions, a failure of the online unit needing corrective repair is possible, then, the corresponding unit in preventive repair interrupts the repair and becomes the online unit.

## References

[1] G. Latouche, V. Ramaswami, Introduction for Matrix Analytic Methods in Stochastic Modeling, ASA-SIAM, 1999.

[2] D. Montoro-Cazorla, R. Pérez-Ocón, A deteriorating two-system with two repair modes and sojourn times phase-type distributed, Reliability Engineering and System Safety 91 (2006) 1–9.

[3] V. Naoumov, Matrix-multiplicative approach to quasi-birth-and-death processes analysis, in: S.R. Chakravarty, A.S. Alfa (Eds.), Matrix-Analytic Methods in Stochastic Models, Marcel Dekker, New York, 1997, pp. 87–106.

[4] M.F. Neuts, Matrix Geometric Solutions in Stochastic Models. An Algorithmic Approach, John Hopkins, University Press, Baltimore, 1981.

[5] F. Neuts, R. Pérez-Ocón, I. Torres-Castro, Repairable models with operating and repair times governed by phase type distributions, Advances in Applied Probability 34 (2000) 468–479.

[6] R. Pérez-Ocón, D. Montoro-Cazorla, A multiple system governed by a quasi-birth-and-death process, Reliability Engineering and System Safety 84 (2004) 187–196.

[7] R. Pérez-Ocón, D. Montoro-Cazorla, Transient analysis of a repairable system, using phase-type distributions and geometric processes, IEEE Transactions on Reliability 53 (2) (2004) 185–192.

[8] R. Pérez-Ocón, D. Montoro-Cazorla, A multiple warm standby system with operational and repair times following phase-type distributions, European Journal of Operational Research 169 (2006) 178–188.

[9] R.H. Yeh, Optimal inspection and replacement policies for multi-state deteriorating systems, European Journal of Operational Research 96 (1996) 248–259.

# Bibliography

[1] A. Aho, M. Lam, R. Sethi, and J. Ullman. *Compilers: Principles, Techniques and Tools*. Addison-Wesley, 1985.

[2] D. A. Bini, B. Meini, S. Steffé, and B. Van Houdt. Structured Markov chains solver: software tools. *Proceedings of SMCtools'06*, 2006.

[3] X.-R. Cao and H.-F. Chen. Perturbation Realization, Potentials, and Sensitivity Analysis of Markov Processes. *IEEE Transactions on Automatic Control*, 42, 1997.

[4] G. Casella, M. Lavine, and C. Robert. Explaining the Perfect Sampler. *The American Statistician*, 55:299–305, 2000.

[5] G. Ciardo and E. Smirni. ETAQA: An Efficient Technique for the Analysis of QBD-processes by Aggregation. *Performance Evaluation, 36-37*, pages 71–93, 1999.

[6] M. Cordy and M.-A. Remiche. QBD sensitivity analysis tool using discrete-event simulation and extension of SMCSolver. In *25th Annual Conference of the Belgian Operations Research Society*, pages 44–45, 2011.

[7] M. Cordy and M.-A. Remiche. QBD sensitivity analysis tool using discrete-event simulation and extension of SMCSolver. In *5th International ICST Conference on Performance Evaluation Methodologies and Tools*, 2011.

[8] S. Dendievel, G. Latouche, and M.-A. Remiche. Perturbation analysis of an M/PH/1 queue. In *Performance 2010, posters*, 2010.

[9] X. K. Dimakos. A Guide to Exact Simulation. *International Statistical Review*, 69:27–48, 2000.

[10] I. S. Gradshteyn and I.M. Ryzhik. *Table of integrals, series and products, 7$^{th}$ edition.* Academic Press, 2007.

[11] S. Hautphenne, K. Leibnitz, and M.-A. Remiche. *Advances in Queueing Theory and Network Applications*, chapter 14, Modeling of P2P File Sharing with a Level-Dependent QBD Process. Springer Science+Business Media, 2009.

[12] B.R. Haverkort, A. P. A. van Moorsel, and A. Dijkstra. MGMtool: A performance analysis tool based on matrix geometric methods. *Modelling Techniques and Tools*, pages 312–316, 1993.

[13] T. Ho ßfeld, K. Leibnitz, and M.-A. Remiche. Exact Sojourn Time Distribution in an Online IPTV Recording System. In *Proceedings of the 15th international conference on Analytical and Stochastic Modeling Techniques and Applications*, pages 158–172. Springer-Verlag, 2008.

[14] R. A. Horn and C. R. Johnson. *Topics in Matrix Analysis.* Cambridge University Press, 1994.

[15] Leonard Kleinrock. Time-shared systems: a theoretical treatment. *J. ACM*, 14:242–261, April 1967.

[16] Leonard Kleinrock and Simon S. Lam. Packet-switching in a slotted satellite channel. In *Proceedings of the June 4-8, 1973, national computer conference and exposition*, AFIPS '73, pages 703–710, New York, NY, USA, 1973. ACM.

[17] D. E. Knuth. *The art of computer programming, volume 2 (3rd ed.): seminumerical algorithms.* Addison-Wesley Longman Publishing Co., Inc., 1997.

[18] G. Latouche and V. Ramaswami. *Introduction to Matrix Analytic Methods in Stochastic Modeling.* Society for Industrial Mathematics, 1999.

[19] L. M. Leemis and S. K. Park. *Discrete-Event Simulation: A First Course.* Prentice Hall, 2006.

[20] J. S. H. Van Leeuwaarden, M. S. Squillante, and E. M. M. Winands. Quasi-Birth-and-Death processes, lattice path counting, and hypergeometric functions. *Applied Probability*, 46:507–520, 2009.

[21] J. S. H. Van Leeuwaarden and E. M. M. Winands. Quasi-birth-death processes with explicit rate matrix. *Stochastic Models*, 22:77–98, 2006.

[22] Q.-L. Li and L. Liu. An Algorithmic Approach for Sensitivity Analysis of Perturbed Quasi-Birth-and-Death Processes. *Queueing Systems*, 48:365–397, 2004.

[23] D. Montoro Cazorla and R. Pérez-Ocón. An LDQBD process under degradation, inspection, and two types of repair. *European Journal of Operational Research*, 190:494–508, 2008.

[24] R. Pérez-Ocón and D. Montoro-Cazorla. A multiple system governed by a quasi-birth-and-death process. *Reliability Engineering and System Safety*, 84:187–196, 2004.

[25] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in Fortran 77: The Art of Scientific Computing.* Cambridge University Press, 2 edition, 1992.

[26] IBM Research. Performance modeling and analysis. `http://www.research.ibm.com/compsci/performance/history.html`, May 2011.

[27] Resnick and Sidney I. *Adventures in stochastic processes.* Birkhauser Verlag, Basel, Switzerland, Switzerland, 1992.

[28] A. Riska and E. Smirni. MAMsolver: A matrix-analytic methods tool. In *12th International Conference on Modelling Tools and Techniques for Computer and Communication System Performance Evaluation*, pages 205–211, 2002.

[29] C. P. Robert and G. Casella. *Monte Carlo Statistical Methods (Springer Texts in Statistics).* Springer-Verlag New York, Inc., 2005.

[30] M. S. Squillante. MAGIC: A computer performance modeling tool based on matrix-geometric techniques. In *Fifth International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, 1991.

[31] E. Thönnes. A Primer in Perfect Simulation. In *Statistical Physics and Spatial Statistics*, pages 349–378. Springer, 2000.

[32] H.C. Tijms. *A First Course in Stochastic Models*, chapter 9, Algorithmic Analysis of Queues. Wiley, Chichester, 2003.